MISSION OPERATIONS AND DATA SYSTEMS DIRECTORATE

Landsat 7 Processing System (LPS) Program Reference Manual

DRAFT

July 1997

National Aeronautics and Space Administration

Goddard Space Flight Center
Greenbelt, Maryland

Landsat 7 Processing System (LPS)
Program Reference Manual
DRAFT

July 1997

Prepared by:

Jeff Hosler, Software Engineering Manager Date
Landsat 7 Processing System, Code 514

Dan Ferry, Software Task Leader Date
Landsat 7 Processing System, CNMOS, Computer Sciences Corp.

Concurred by:

Nate Daniel, Element Manager Date
Landsat 7 Processing System, CNMOS, Computer Sciences Corp.

Approved by:

Joy Henegar, Project Manager Date
Landsat 7 Processing System, Code 514

Goddard Space Flight Center
Greenbelt, Maryland

Abstract

This document provides information necessary for Earth Resources Observation System (EROS) Data Center (EDC) programmers to maintain and enhance the Landsat 7 Processing System (LPS).

Keywords: *Landsat 7, Landsat 7 Processing System (LPS), maintenance*

Preface

This document is under the control of the Landsat 7 Processing System (LPS) Central Review Board (CRB).

Configuration change requests (CCRs) to this document, as well as supportive material justifying the proposed changes, should be submitted to the LPS CRB. Changes to this document shall be made by document change notice (DCN) or by complete revision.

Address questions and proposed changes concerning this document to

Jeffrey C. Hosler, Software Manager
Landsat 7 Processing System
Code 514
Goddard Space Flight Center
Greenbelt, Maryland 20771

Contents

Section 5. Portability Issues

Section 6. Development Environment

Section 7. Testing LPS Software

Section 8. Design Decisions

Section 1. Introduction

1.1 Purpose and Scope

This document provides information necessary for Earth Resources Observation System (EROS) Data Center (EDC) programmers to maintain and enhance the Landsat 7 Processing System (LPS). This document is not intended to be a design document, users guide, or configuration guide. Instead, it contains additional information that was not appropriate for the other documents, but that the LPS development team considered useful for maintenance programmers.

This document covers the following topics with respect to maintaining LPS software:

- Commercial off-the-shelf (COTS) software
- Government off-the-shelf (GOTS) software
- Reuse software
- Portability issues
- Development environment
- Testing LPS software outside of the full environment
- Design decisions made during development
- Other "gotchas"

1.2 Applicable Documents

LPS maintenance programmers should be familiar with the following documents before reading this document:

1. National Aeronautics and Space Administration (NASA) Goddard Spaceflight Center (GSFC), Mission Operations and Data System Directorate (MO&DSD), 560-8SWR/0195, *Landsat 7 Processing System (LPS) Software Requirements Specification (SRS)*, Revision 2, June 1997
2. —, *Landsat 7 Processing System (LPS) As-Built Specification (ABS)*, July 1997
3. —, *Landsat 7 Processing System (LPS) Interface Definitions Document (IDD)*, July 1997
4. —, 560-3SUG/0195R2.0, *Landsat 7 Processing System (LPS) Users Guide*, Release 2, Volumes 1 and 2, July 1997

Section 2. COTS Software

Three COTS software components are used by LPS: ORACLE, the high-speed peripheral device interface (HPDI) data capture board driver, and the hierarchical data format (HDF) library. ORACLE products were not modified in any way for use with LPS. However, the HPDI data capture board driver software was written by Silicon Graphics, Inc. (SGI) engineers specifically to support LPS.

2.1 ORACLE

Important items to consider when maintaining LPS ORACLE routines are the following:

- User interface routines were generated with Developer 2000 Forms 4.5. The .fmb files are converted by f45genm to .fmx files.
- Three reports are generated using Developer 2000 Reports 2.5 (the specific units are mac_ui_data_trans_sum_rpt, mac_ui_lps_qa_rpt, and mac_ui_data_rec_sum_rpt). The .rdf files are converted by r25convm to .rep files.
- Global and subsystem-specific database access routines (DBARs) hide the details of accessing the database. These routines have names such as subsystem_db_GetSomeData.pc and are processed by the precompiler into .c routines. (This is handled by Imakefiles.) Include files (e.g., subsystem_db_GetSomeData.h) are also sometimes used.

2.2 HPDI Device Driver

The HPDI device driver software was designed and developed by SGI. This software is not included with the LPS delivered source, but is treated as COTS software, and is required to perform raw wideband data capture and transmission. If changes to this software are required, contact SGI Technical Support.

2.3 Hierarchical Data Format Library

HDF is a product of the National Center for Supercomputing Applications (NCSA) at the University of Illinois. It is a platform-independent file format designed to contain scientific data. Data objects of differing types (e.g., multidimensional arrays, tables, images, text) can be stored as distinct or related entities in the same file. HDF files are created and read using an automatic programmatic interface (API) implemented by a library of C functions available free of charge from NCSA. HDF is available for most UNIX platforms, as well as Windows NT and the Apple Macintosh. HDF files created on one platform can be transported to any of the other supported platforms, and a number of third-party tools are available that can read HDF files. The LPS browse and mirror scan correction data (MSCD) files are HDF files. NCSA's Web site at http://hdf.ncsa.uiuc.edu/ provides details about HDF and compatible tools.

Section 3. GOTS Software

Three software components are provided by NASA/GSFC. If any changes to the software are needed, contact GSFC for their support.

3.1 Waveletting

GSFC provided the C function, wavelet_alg.c, to the LPS project to perform wavelet reduction on browse images. LPS developers had to make some minor modifications to this function so that it could be integrated into LPS, but they did not change the logic of the algorithm. The wavelet_alg.c function receives a pixel array as input, performs wavelet reduction on it, and outputs a smaller array containing a reduced representation of the input image. The degree of reduction is determined by the number of wavelet iterations performed. This function is used by the idp_browse component of the image data processing subsystem (IDPS).

3.2 Frame Synchronization

LPS uses GSFC's Renaissance Consultative Committee for Space Data Systems (CCSDS) frame synchronization software building block as part of the

- Raw data processing program to locate CCSDS frames within a contact's file of raw wideband data
- Payload correction data (PCD) program to locate the start of PCD minor frames within sequences of packed PCD words

The CCSDS frame synchronization software locates fixed-length frames within sequences of octets stored in memory. The software uses a parameterized search, check, lock, and flywheel (SCLF) synchronization strategy that follows the state transition diagram (Figure 3–1).

Frame synchronization using an SCLF strategy is always in one of four modes:

1. **SEARCH** – The software begins in this mode and searches bitwise for the frame synchronization pattern in the input buffer. Once the pattern is discovered, the software moves to CHECK.
2. **CHECK** – The software verifies that it has found the pattern by checking that the pattern continues to occur at the expected location a parameterized number of times. If the pattern recurs the requisite number of times, the software moves to LOCK. Otherwise, it returns to SEARCH.
3. **LOCK** – The software outputs frames as long as it continues to find the pattern at the expected location. After each discovery of the pattern, the software advances the length of the frame and looks for the synchronization pattern. If the pattern is not encountered, the software may either enter FLYWHEEL or return immediately to SEARCH.
4. **FLYWHEEL** – The software flywheels, i.e., ignores a parameterized number of missing patterns. If the pattern is discovered again before the specified number of missing patterns is exceeded, the software returns to LOCK. Otherwise, it returns to SEARCH. Note that, if the FLYWHEEL parameter is 0, the software goes immediately from LOCK to SEARCH when the pattern is missing.

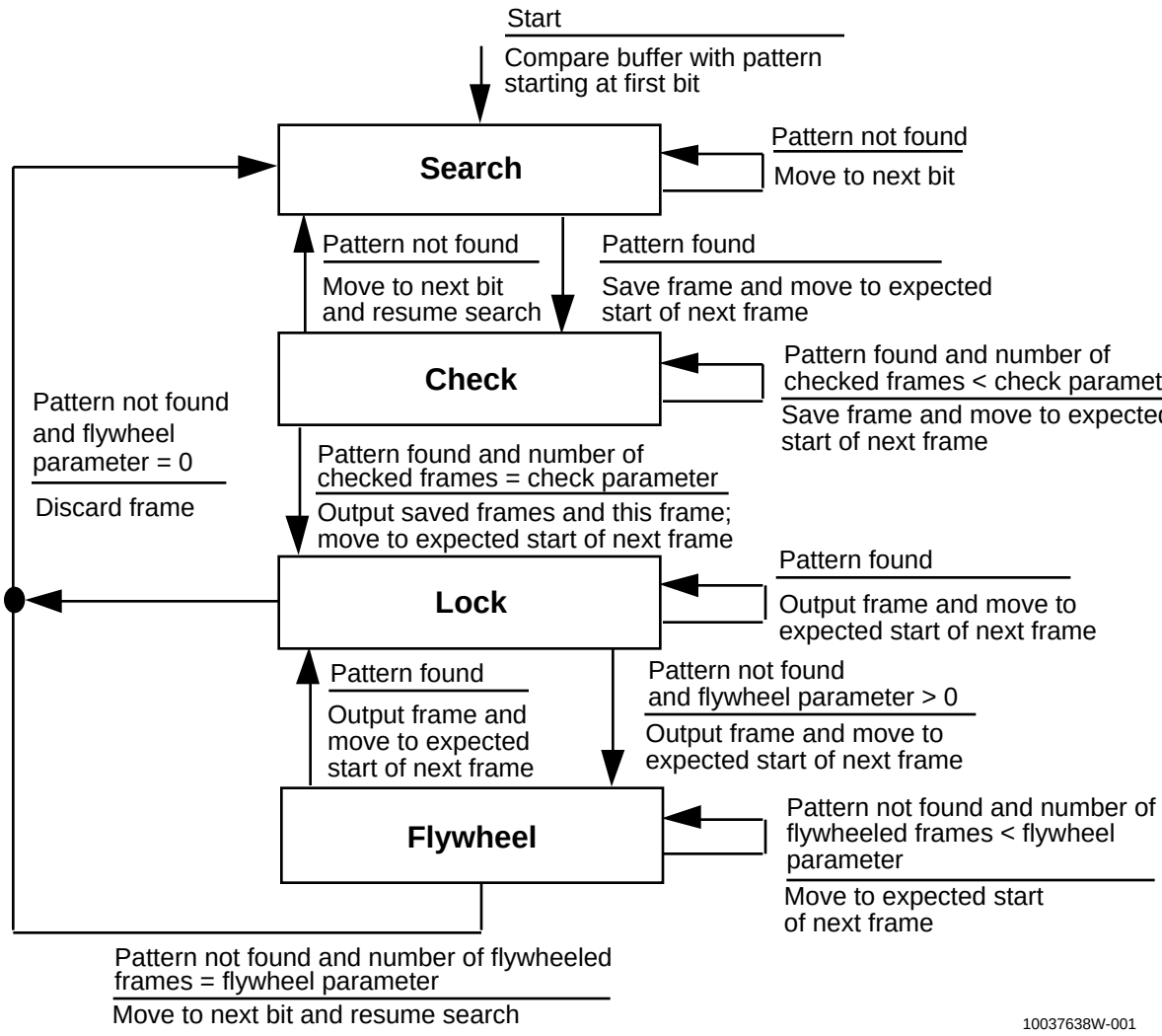

Figure 3–1. State Transition Diagram for Frame Synchronization Using an SCLF Strategy

A set of parameters provided by the caller controls details of the synchronization strategy. In addition to the flywheel parameter, there are parameters for the number of frames to check, the number of bit errors in the synchronization pattern to ignore, a number of bit positions by which the pattern can be offset and still count as successfully located, and others.

To invoke the frame synchronization software, the using program first calls fs_initialize(), passing a structure containing the parameter values to be used and the address of a callback function. The using program loops through its input data, coping a convenient portion to an input buffer in memory and calling fs_frame_sync() with a pointer to the buffer. fs_frame_sync() locates frames and invokes the callback function for each frame that it locates, passing a pointer to the frame along with other information. fs_frame_sync does not return until the entire buffer has been processed. When all input has been processed, the using program calls fs_terminate to free resources allocated during synchronization and to generate quality and accounting (Q&A)

statistics for the entire processing run. The using program retrieves the statistics by calling fs_get_stats().

3.3 HDF Earth Observation System Library

The HDF Earth Observation System (EOS) library is a set of functions, developed by the EOS Core System (ECS) project, that implements the HDF-EOS API. These library functions can be called from C/C++ programs and are built on the base HDF library from NCSA. HDF-EOS uses base HDF functions to build higher level EOS-specific structures, such as swaths and geolocation data.

An HDF-EOS swath is a structure consisting of a track dimension and a cross-track dimension. The track represents the path of a spacecraft as projected on the Earth's surface. In the case of LPS, the cross-track represents the path of the mirror on the spacecraft as it sweeps back and forth perpendicular to the track. The result is a swath consisting of a two-dimensional array of sensor data.

Geolocation data is HDF-EOS structures that contain information about the swath and are internally mapped to it. Data, such as scan times and latitude and longitude coordinates, is stored as geolocation data and mapped to scan data in the swath. Through the HDF-EOS API, an application can extract sections of scan data from the swath based on geolocation data. HDF-EOS files are HDF files also; however, most off-the-shelf HDF-compatible tools cannot interpret the complex EOS swath and geolocation structures.

It is important to note that HDF and HDF-EOS files are not completely compatible. The LPS band and calibration files are written as swaths and geolocation data using the HDF-EOS API.

3.4 Reusable Software Library

The Reusable Software Library (RSL) is a collection of general-purpose mathematical functions in C programming language. The PCD processing subsystem (PCDS) uses the following functions from the RSL:

- minv3c – invert a 3 x 3 matrix
- mprodgc – compute the cross product of two matrixes
- qtoac – compute a 3 x 3 rotation matrix from a quaternion

A full description of the API for each function is included in the source file's prolog.

Section 4. LPS Global Units

4.1 Finding Routines that Made Calls to a Given Routine

LPS has a script named “Whocalls” that can be used to determine which routines made calls to a routine(s) within the scope of LPS project. This script searches through all the directories whose names began with “src” and under the $LPS_HOME directory and reports those routines that made calls to that routine(s).

4.2 LPS Global Units

LPS global routines belong to the following categories:

1. LPS shared-memory resource management
2. LPS first-in-first-out (FIFO) queue
3. LPS file-related operation
4. LPS process status, initialization, and handling
5. LPS message logging
6. LPS time manipulation
7. LPS database access

4.2.1 LPS Shared-Memory Resource Management

- **lps_ShmAddListTail** – adds a block into the tail of a shared-memory block list
- **lps_RsrcAlloc** – creates the interprocess communication (IPC) resources used by LPS
- **lps_RsrcAllocFIFO** – creates the FIFO queues used by LPS
- **lps_RsrcAllocShm** – creates the shared-memory segments used by LPS
- **lps_ShmClose** – allows the caller to detach from an LPS shared-memory resource
- **lps_ShmCreate** – creates the shared-memory resources used by LPS
- **lps_ShmCreateSemaphore** – creates an LPS semaphore set
- **lps_RsrcDealloc** – deallocates LPS IPC resources
- **lps_ShmGetRdBlk** – provides the caller with a shared-memory read block
- **lps_ShmGetWrBlk** – provides the caller with a shared-memory write block
- **lps_ShmOpen** – attaches the caller to an LPS shared-memory resource
- **lps_ShmOpenSemaphore** – attaches the caller to an LPS semaphore set
- **lps_ShmPutRdBlk** – returns a read block to the free shared-memory block pool

- **lps_ShmPutWrBlk** – places a written block to the shared-memory active block pool
- **lps_ShmRemListHead** – removes a block from the head of the active shared-memory block list
- **lps_ShmRemListTail** – removes a block from the tail of the free shared-memory block list.
- **lps_ShmRemove** – removes an LPS shared-memory resource from the system

4.2.2 LPS FIFO Queue

- **lps_FIFOClose** – detaches the caller from the requested LPS FIFO queue channel
- **lps_FIFOOpen** – attaches the caller to the requested IPC FIFO channel
- **lps_FIFOSend** – sends data into the LPS IPC FIFO channel
- **lps_FIFOReceive** – retrieves data from LPS IPC FIFO channel

4.2.3 LPS File-Related Operation

- **lps_GetPIDFileName** – obtains the temporary process identifier (PID) filename used to signify raw data capture subsystem (RDCS) activity
- **lps_FileNameCreate** – creates an LPS L0R output filename with a full path based on the file type and subinterval identifier
- **lps_ParseFileName** – retrieves the file pathname, filename, and file type
- **lps_ValidateRDCOutfileName** – parses and validates the raw data capture filename from command line input argument

4.2.4 LPS Process Status, Initialization, and Handling

- **lps_CaptureIsRunning** – checks to see if raw data capture process is running
- **lps_GetOpt** – extracts the option letter from argument vector argv
- **lps_ParseOptions** – parses and validates the arguments for LPS processes according to the LPS Interface Definition Document (IDD), Section 2.1
- **lps_ProcessChildStatus** – converts the child's system status to an LPS-specific child exit status
- **lps_ProcessInit** – initializes the processing environment for LPS processes
- **lps_ProcessStartChild** – forks a child process, executes the requested file or script, and passes to the child a specified argv vector list

4.2.5 LPS Message Logging

- **lps_LogMessage** – logs lps error or status messages into lps_Jounal using a UNIX syslog facility

4.2.6 LPS Time Manipulation

- **lps_CalDate** – computes the calendar date corresponding to a given Julian day
- **lps_ComputeEpoch** – converts time from structure to time in seconds from Epoch
- **lps_DayOfYear** – computes the day of year from a given month, day, and year
- **lps_EpochBreakdown** – converts time in seconds from Epoch to LPS time structure
- **lps_JulDate** – computes the Julian day corresponding to a given calendar date
- **lps_MonthDay** – computes month and day of month from a given day of year and year
- **lps_TimeAdd** – adds a time duration to a time value
- **lps_TimeAssignValue2Struct**. – assigns values to an LPS time structure
- **lps_TimeCompare** – compares two time values and passes back an integer greater than, equal to, or less than 0 based on first time value greater than, equal to, or less than second time value
- **lps_TimeCompareTol** – compares the difference between two time values against a given tolerance
- l**ps_TimeDiff** – compares two time values and returns the difference and comparison status
- **lps_TimeDivide** – calculates the total number of time units for a time duration
- **lps_TimeDurCheck** – checks the validity of members of an LPS time-duration structure
- **lps_TimeGetCurrentTime** – returns system time in yyyy-mm-dd-hh-mm-ss format
- **lps_TimeMultiply** – calculates a time duration given total number of time units and base time unit structure
- **lps_TimeScanSeconds** – converts a scan time from time structure format to a time in seconds since 1/1/1993
- **lps_TimeString2Struct** – converts a time string into a time structure
- **lps_TimeStringCheck** – checks the validity of an LPS time string
- **lps_TimeStruct2String** – converts a time in LPS structure into a time string in the format of LPS time structure

- **lps_TimeStructCheck** – checks validity of members of an LPS time structure
- **lps_TimeSubtract** – subtracts a time in LPS time structure from a duration in LPS duration structure and returns a result in LPS time structure

4.2.7 LPS Database Access

- **lps_db_Commit** – commits the database transactions and reports the commit status
- **lps_db_Connect** – connects a process to the database server
- **lps_db_Disconnect** – disconnects a process from the database server
- **lps_db_ErrorMessage** – logs an ORACLE error message
- **lps_db_GetLPSConfiguration** – retrieves the LPS configuration table from the database server
- **lps_db_GetRDCInfo** – retrieves raw data capture information in the RDC_ACCT table from the database server
- **lps_db_GetSubIntvInfo** – retrieves the subinterval information table from the database server
- **lps_db_InsertFileInfo** – inserts L0R file information into the LPS database
- **lps_db_Rollback** – rolls back the database transactions and reports the rollback status

4.3 MACS Global Units

Management and control subsystem (MACS) global routines are as follows:

- **mac_sems** – contains two functions: psem and vsem. Psem decrements a passed semaphore, and vsem increments a passed semaphore. It is used by the mac_auto_archive and mac_auto_startL0R processes for resource sharing.

4.4 RDCS Global Units

RDCS global routines are as follows:

- **rdc_CheckDiskSpace** – Checks and reports on the disk space of a specified path. This function has two arguments: path and thresholdMB. *path* defines the path where the disk space is to be checked; *thresholdMB* defines a threshold in megabytes used to report a warning.
- **rdc_DeIsolateProcess** – Attempts to deisolate and unrestrict the specified processor, and sets the process to a normal priority. This function has one argument: processorNum. *processorNum* defines the processor to attempt to unrestrict. This function should always be called before terminating the process if rdc_IsolateProcess was used to restrict the processor. See: rdc_IsolateProcess.

- **rdc_DeviceFunctions** – Encapsulates all HPDI device-specific routines. This unit contains some code written by SGI to properly communicate HPDI with the device driver also written by SGI. This unit does not contain the device driver software.

- **rdc_FileSplit** – Splits a specified path into its path and file components. This function has three arguments: inpath, path, and file. *inpath* contains the path to split. *path* and *file* will be returned to the caller containing the *path* portion of *inpath* and the *file* portion of *inpath*, respectively.

- **rdc_GetBinNumber** – Returns the currently scheduled digital linear tape (DLT) BIN number for tape archival. This function has two arguments: tapeLibBinFile and binNum. *tapeLibBinFile* contains the name of the file to be used for reading the scheduled DLT BIN number; *binNum* will be returned containing the currently scheduled DLT BIN number. This function expects a *tapeLibBinFile* that contains a single American Standard Code for Information Interchange (ASCII) integer that defines the DLT BIN number currently scheduled for tape archival. If the specified *tapeLibBinFile* file does not exist, failure is returned. See rdc_SetBinNumber and rdc_TapeBinCount.

- **rdc_GetEnvironment** – Loads the values of the specified environment variables into the variables specified in the provided structure. This function has two arguments: numEnvVars and envVars. *numEnvVars* defines the number of environment variable records contained within the *envVars* structure. Contained within the rdc.h/rdc_extern.h header files is a macro called NUMBER that should be used to set *numEnvVars*. Also contained within these header files is the structure definition rdcEnvStruct, which defines the format of *envVars*. The *envVars* structure should be defined with “static char *” variables, where this function will assign the values of the specified environment variables. If the environment variables are not defined on the system, the static variables will be set to the specified default values. For example:

```
static char *rawPath;
static char *tempPath;
rdcEnvStruct envVars[] = {
 {“LPS_RAWFILE_PATH”, (char **) &rawPath, “.”},
 {“LPS_TEMFILE_PATH”, (char **) &tempPath, “.”},
};
numEnvVars = NUMBER(envVars);
```

  Specifies that two environment variables, LPS_RAWFILE_PATH and LPS_TEMFILE_PATH, are to be extracted from the system and the values should be assigned to the rawPath and tempPath static variables, respectively. If the values are not defined on the system, the default values for both will be the current directory (“.”).

  If *numEnvVars* is set to 0, this function does nothing.

This function is called by the rdc_Init function or can be called independently by the application. See: rdc_Init.

- **rdc_Init** – Designed to be used by all RDCS applications to eliminate repetitive initialization source code. This function performs four tasks.

  1. Initializes two global variables: rdc_host and rdc_user; rdc_host will be set to the host string name (e.g., lps001) and rdc_user will be set to the user's login ID (e.g., root)
  2. Calls rdc_GetEnvironment to extract the system environment variables needed by the application
  3. Calls rdc_ParseCommand to parse the command line arguments passed to the application (argc, argv)
  4. Calls lps_ProcessInit to initialize the process in the same manner as all LPS executables, which optionally sets up a signal handler, connects to the database, and checks for a running capture process

  This function has eight arguments: argc, argv, SigHandler, numOptions, options, numEnvVars, envVars, and Connect. *argc* and *argv* should be provided as passed to the application. *SigHandler* defines the signal handler to set up to catch general termination signals. *numOptions* and *options* should be provided as specified in the description for rdc_ParseCommand. *numEnvVars* and *envVars* should be provided as specified in the description for rdc_GetEnvironment. *Connect* is a Boolean flag indicating whether a connection should be made to the LPS database. See lps_ProcessInit, rdc_GetEnvironment, and rdc_ParseCommand.

- **rdc_IsolateProcess** – Attempts to isolate and restrict the specified processor and sets the process to the highest nondegrading priority. This function has one argument: processorNum. *processorNum* defines the processor to attempt to restrict. See rdc_DeIoslateProcess.

- **rdc_LoadTape** – Loads a tape from the specified DLT BIN. This function has two arguments: device and binNum. *device* specifies the DLT robotic arm device, and *binNum* specifies the DLT BIN number where the tape is located. See rdc_UnloadTape.

- **rdc_LogFileError** – Provides a means of logging a meaningful interpretation of the system errno to the LPS Journal when a file access or input/output (I/O) error has occurred. This function has two arguments: description and filename. *description* should be passed as a meaningful description of the file that produced the error (e.g., Capture Accounting). *filename* contains the actual filename of the file that produced the error. If the filename is not known, a NULL value should be passed. This function currently there handles a limited number of errno values. This function was designed to allow for future expansion of additional errno values.

- **rdc_LogShutdownMessage** – Provides all RDCS processes with a means to log a standard shutdown message given a termination status. This function has one argument:

status. *status* defines the process RETURN_CODE that is being reported on shutdown of the process.

- **rdc_ParseCommand** – Loads the values of the command line arguments into the variables specified in the provided structure. This function has four arguments: argc_in, argv_in, numOptions, and options. *argc_in* and *argv_in* should be provided as passed to the application "main," or should be in the same format. *numOptions* defines the number of command line argument records contained within the *options* structure. Contained within the rdc.h/rdc_extern.h header files is a macro called NUMBER that should be used to set *numOptions*. Also contained within these header files is the rdcOptionStruct structure definition, which defines the format of *options*. The *options* structure should be defined with static variables, where this function will place the values of the specified command line arguments. Adequate memory allocation must exist for any destination arrays If the command line arguments do not exist on the command line, the static variables will be returned unchanged.

  Three different command line argument types are defined for RDCS. These values are specified in an enumerated type contained within the rdc.h/rdc_extern.h header files called rdcOptionType. The possible values are rdcArg, rdcNoArg, and rdcFile. rdcArg indicates that the command line option requires an argument. rdcNoArg indicates that the command line option is a Boolean flag (provided or not provided). rdcFile indicates that the final command line argument is a filename (no option), e.g., rdc_Save filename. For example,

```
"static Boolean flag = FALSE;"
static char argument[512];
static char file[512];
rdcEnvStruct options[] = {
 {"-x", rdcArg, (void *) &argument, 512},
 {"-y", rdcNoArg, (void *) &flag, NULL},
 {"(char*NULL)", rdcFile, (void *) &file, 512},
};
numOptions = NUMBER(options);
```

  specifies that three command line arguments are allowed to be passed to this application, -x, -y, and (char*NULL), and the values should be assigned to the argument, flag, and file static variables, respectively. The first structure record element defines the allowed option. The second record element defines the type of option. The third record element specifies the destination variable to hold the value. The last record element defines the maximum allowed length of the argument or file (the destination array size). If the command line arguments are not provided, then the values for the static variables will be unchanged.

If *numOptions* is set to 0, this function does nothing.

This function is called by the rdc_Init function or can be called independently by the application. See rdc_Init.

- **rdc_ResumeProcess** – Sends a resume signal (SIGCONT) to the process group leader defined by the #define RDC_L0R_GROUPLEADER. This #define is contained with the global RDCS header files rdc.h/rdc_extern.h. This function attempts to resume execution of all the processes running in the same group as the group leader. Processing should be suspended with a call to rdc_SuspendProcess. It should be noted that for SIGCONT to work, the process calling this function must be run as "root." This can be accomplished two ways:

  1. The user could be logged in as "root" when the process is executed.
  2. The executable must have the Set Group ID bit set in the permissions, and the executable must be owned by "root." This is the preferred option.

  Refer to chmod(1) for details. The second method is used within LPS. See rdc_SuspendProcess.

- **rdc_SetBinNumber** – Sets the next scheduled DLT BIN number for tape archival. This function has two arguments: tapeLibBinFile and binNum. *tapeLibBinFile* contains the name of the file to be used for writing the next scheduled DLT BIN number, *binNum* contains the next scheduled DLT BIN number. If *tapeLibBinFile* does not exist, one is created prior to exiting. See rdc_GetBinNumber and rdc_TapeBinCount.

- **rdc_ShutDown** – Two philosophies for signal handling exist within LPS. First, the signal handler cleans up the application and exits. Processing is halted, and the signal handler does not return to the application for continued processing. Second, the signal handler simply sets a global variable indicating that a signal was received and returns to the application for continued processing. This function subscribes to the first method. This signal handler performs three functions:

  1. A global function pointer variable is called (if set) to clean up the application. This global variable is defined in the rdc.h/rdc_extern.h header files and is called rdc_CleanupFPtr.
  2. rdc_LogShutdownMessage is called to log an appropriate shutdown message.
  3. The function calls exit(2) to exit the application with an appropriate termination status.

  This function has one argument: sig. *sig* is provided by the Kernel indicating the signal that caused the processing interruption. This function could be, but should not be, called directly by an application. It is intended to be solely for signal handling. See rdc_LogShutdownMessage.

- **rdc_SuspendProcess** – Sends a suspend signal (SIGSTOP) to the process group leader defined by the #define RDC_L0R_GROUPLEADER. This #define is contained with

the global RDCS header files rdc.h/rdc_extern.h. This function attempts to suspend, or put to sleep, all the processes running in the same group as the group leader. Resumption of processing should be performed with a call to rdc_ResumeProcess. It should be noted that for SIGSTOP to work, the process calling this function must be run as "root." This can be accomplished two ways

1. The user could be logged in as root when the process is executed.
2. The executable must have the Set Group ID bit set in the permissions, and the executable must be owned by "root." This is the preferred option.

Refer to chmod(1) for details. The second method is used within LPS. See rdc_ResumeProcess.

- **rdc_System** – "Mirrors" the UNIX system(2) command with the one exception. system(2) returns a success or failure depending on whether is successfully executed the specified process. rdc_System has the same behavior, except that it returns the return status of the specified process when it has a successful fork/exec of the process. This means that even with a successful fork/execution of the process, the rdc_System function could still return a failure status. This does not happen with the UNIX system(2). This function has one argument: cmd. *cmd* defines the command that requires execution. *cmd* should be provided as defined in system(2). See rdc_SystemMonitor.

- **rdc_SystemMonitor** – Works similarly to the rdc_System command, except that the return is immediate. This function does not wait for the completion of the forked/executed process, but is designed to be called repetitively until the process terminates. This allows the caller to monitor the forked process. Function return values indicate the current state of the forked/executed process. This function adds an argument to the rdc_System definition: pid. *pid* will be returned to the caller containing the process ID of the forked/executed process. Repetitive calls to this function require that *pid* be passed for each call. This first time this function is called, *pid* will be set. Successive calls will reference the value in *pid*. See rdc_System.

  NOTE: Only one process can be monitored at a time.

- **rdc_TapeBinCount** – Determines the number of DLT BINs available on the specified DLT device. This function has three arguments: binCount, device, and tmpDir. *binCount* will be returned containing the total number of DLT BINs available on the specified *device*. *tmpDir* specifies the temporary directory to be used for generating internal temporary files. See rdc_GetBinNumber and rdc_SetBinNumber.

- **rdc_TermSig** – Two philosophies for signal handling exist within LPS. First, the signal handler cleans up the application and exits. Processing is halted, and the signal handler does not return to the application for continued processing. Second, the signal handler simply sets a global variable indicating that a signal was received, and returns to the application for continued processing. This function subscribes to the second method. This signal handler performs one function. If the global variable global_pid, defined in rdc.h/rdc_extern.h, is set, this function attempts to terminate the process specified by the

variable. This function has one argument: *sig*. *sig* is provided by the Kernel indicating the signal that caused the processing interruption. This function is only used by the RDCS Restage process, but should be enhanced for future use by the RDCS Save process, which would eliminate the rdc_SaveSignalHandler function.

- **rdc_TimeDiff** – Returns the difference in seconds of two LPS time structures (lps_TimeStruct). This function has two arguments: time1 and time2. *time1* and *time2* contain the LPS time structures used to perform the calculation. This function will return the absolute value of the difference in seconds (ordering does not matter) or a #define RDC_TIMEDIFF_ERROR if failure occurs.

- **rdc_TimeString2Struct** – Converts an RDCS time string (YYDDDHHMMSS) to an LPS time structure (lps_TimeStruct). This function has two arguments: timeString and timeStruct. *timeString* contains the RDCS time string in the format YYDDDHHMMSS; *timeStruct* will be returned containing the LPS time structure corresponding to *timeString*. See rdc_TimeStruct2String.

- **rdc_TimeStringValid** – Validates an RDCS time string. This function has one argument: timeString. *timeString* should be in YYDDDHHMMSS format. A value of TRUE is returned if the *timeString* is valid and a value of FALSE is returned if it is not.

- **rdc_TimeStruct2String** – Converts an LPS time structure (lps_TimeStruct) to an RDCS time string (YYDDDHHMMSS). This function has two arguments: timeStruct and timeString. *timeStruct* contains the LPS time structure; *timeString* will be returned to contain the corresponding RDCS time string in YYDDDHHMMSS format. See rdc_TimeString2Struct.

- **rdc_UnloadTape** – Unloads a tape from the specified DLT BIN. This function has two arguments: device and binNum. *device* specifies the DLT robotic arm device, and *binNum* specifies the DLT BIN number where the loaded tape is to reside. See rdc_LoadTape.

- **rdc_db_DeleteRDCFiles** – Performs deletion of the raw wideband data file and associated accounting file. This DBAR requires two arguments: rdc_deleteFileNameArg and rdc_unconditionalDelete. This DBAR will query the RDC_ACCT database table to establish that the specified raw wideband data file (rdc_deleteFileNameArg) has been successfully archived to tape and L0R processed to completion. If the raw wideband data meets these criteria, the files are deleted. If the raw wideband data does not meet these criteria, the files are left on the system and a message is generated. The database record validation may be overridden with the rdc_unconditionalDelete flag set to TRUE.

- **rdc_db_FileExistence** – Establishes whether the specified raw wideband data file has an associated record in the RDC_ACCT database table. This DBAR has two arguments: rawFilename and num. *rawFilename* is used as the RDC_ACCT database table key, and *num* will be returned to the caller containing the number of records found that contain *rawFilename*.

- **rdc_db_LoadLabelParms** – Loads the specified structure with tape label information, extracted from the RDC_ACCT database table, and associated with a particular raw wideband filename. This DBAR has two arguments: rawFilename and rdc_LabelArgs. *rawFilename* is used as the key to perform the database query. *rdc_LabelArgs* will be returned to the caller containing the tape label information extracted from the queried record. The tape label structure definition is contained within the header files rdc_GenLabel.h and rdc_GenLabelExtern.h and is called rdcLabelStruct.

  NOTE: This unit is unconventional in the sense that it performs a database connect and disconnect instead of expecting the database connection to be performed by the caller. All other RDCS DBARs leave the connection to the caller. For convention, this DBAR should eventually be modified to remove the database connection and disconnection. In addition, to maintain a consistency of treating the DBARs as global functions, the rdcLabelStruct definition should be moved to the rdc.h and rdc_extern.h header files and removed from the existing location.

- **rdc_db_RegisterProcess** – Registers or unregisters the current process into/from the PROCESS_ID database table. This DBAR has two arguments: processName and regType. *processName* defines the name that the calling process wants to register into the database, e.g., RDC_SAVE. *regType* indicates whether the process is registering or unregistering. *regType* is defined as the enumerated rdcRegisterType defined in the rdc.h and rdc_extern.h header files.

  NOTE: Two units currently exist for the MACS that performs this same function: mac_db_RegL0RPID.pc and mac_db_UnregL0RPID.pc. Unfortunately, these units were written specifically for L0R processing and could not be reused in the RDCS as written. This unit was written more generically to handle any process registration and could, eventually, be moved to the LPS globals and integrated into the MACS. This would potentially eliminate unneeded source.

- **rdc_db_SetArchiveFlag** – Sets the ARCHIVE_FLAG in the RDC_ACCT database table for the specified raw wideband data filename. This DBAR has two arguments: rdc_Filename and rdc_ArchiveFlag. *rdc_Filename* specifies the raw wideband data filename to be used as the query key into the RDC_ACCT database table. *rdc_ArchiveFlag* is a Boolean flag indicating the archival state of the raw wideband data filename (*rdc_Filename*).

  NOTE: *rdc_ArchiveFlag* currently is defined as an int. This is not incorrect, but for consistency, should be changed to Boolean.

- **rdc_db_SetOnLineFlag** – Used to bring the LPS database to a consistent state by setting the ON_LINE_FLAG in the RDC_ACCT database table to indicate which raw wideband data files are currently online. This DBAR has no arguments. Initially, this unit sets all RDC_ACCT records to indicate an offline status. It then proceeds to identify the raw wideband data files that are online, and sets the associated RDC_ACCT record to indicate that the file is online.

NOTE: At this time, this unit does not verify the existence of both the raw wideband data file and the associated accounting file. It only checks for the raw wideband data file. This unit should be enhanced to also check for the associated accounting file.

- **rdc_db_WriteAcctToDb** – Given a raw wideband data accounting file, inserts a new record into the RDC_ACCT database table associated with the raw wideband data. This DBAR has two arguments: RDCAcctFD and RDCOnlineFlag. *RDCAcctFD* is a file descriptor for an opened raw wideband data accounting file. *RDCOnlineFlag* is a Boolean flag indicating the online state of the raw wideband data.

  NOTE: *RDCOnlineFlag* currently is defined as an int. This is not incorrect, but for consistency, should be changed to Boolean.

- **rdc_db_WriteOnLineFlag** – Sets the ON_LINE_FLAG in the RDC_ACCT database table for the specified raw wideband data filename. This DBAR has two arguments: rdc_Filename and rdc_OnLineFlag. *rdc_Filename* specifies the raw wideband data filename to be used as the query key into the RDC_ACCT database table. *rdc_OnLineFlag* is a Boolean flag indicating the online state of the raw wideband data filename (*rdc_Filename*).

  NOTE: *rdc_OnLineFlag* currently is defined as an int. This is not incorrect, but for consistency, should be changed to Boolean.

4.5 RDPS Global Units

Raw data processing subsystem (RDPS) global routines belong to the following categories:

- Frame synchronization
- Cyclic redundancy check (CRC)
- Reed-Solomon (RS) decoder
- RS code block
- Bose-Chaudhuri-Hocquenghem (BCH) decoder

4.5.1 Frame Synchronization

- **fs_align_frames_n_output** – aligns frames and calls output module
- **fs_frame_synchronization** – locates and byte synchronizes frames in raw data
- **fs_get_stats** – retrieves statistics of frame synchronization process (FSP) so far
- **fs_initialize** – initializes FSP
- **fs_match_fsp** – checks if FSP exists at bit location
- **fs_match_fsp_slip** – checks if FSP found with range of bits
- **fs_pn_decode** – CCSDS pseudorandom noise decodes the frame

- **fs_reverse_bits** – reverses bits in a frame
- **fs_terminate** – terminates FSP
- **fs.h** – defines tolerance of SCLF mode and parameters used in frame synchronization

4.5.2 Cyclic Redundancy Check

- **rdp_CRCGenTable** – generates table for CRC checksum calculation
- **rdp_CRCCheckSum** – performs CRC checksum calculation

4.5.3 Reed-Solomon Decoder

- **rdp_RSD** – deinterleaves the data, performs RS decoding on each encoded data frame

4.5.4 Reed-Solomon Code Block

- **rsd.h** – defines RS decoder parameters and how to set them to decode RS codes
- **gf16.gft** – generates conventional table

4.5.6 BCH Decoder

- **rdp_BCHBuildMsnQuadTable** – reads in Mission Quad table from rdp_BCHQuadFile
- **rdp_BCHCreateTransTable** – generates BCH table for deinterleave
- **rdp_BCHMakeMsnSyndromeTable** – calculates syndromes for mission codeblocks
- **rdp_BCHMsnCalcSyndromes** – calculates syndromes for mission codeblocks
- **rdp_BCHMsnChienSearch** – finds roots to error locator polynomial for mission codeblocks
- **rdp_BCHMsnDecTree** – creates mission locator polynomial for Chien search
- **rdp_BCHMsnDivide** – divides one element of a Galois field (mission field) by another
- **rdp_BCHPtrCalcSyndromes** – calculates syndromes for pointer codeblocks
- **rdp_BCHPtrChienSearch** – finds the roots to error locator polynomial for pointer codeblocks
- **rdp_BCHPtrDecTree** – creates pointer locator polynomial for Chien search
- **rdp_BCHPtrDivide** – divides one element of Galois field (pointer field) by another
- **rdp_BCHTransposeCadu** – transposes channel access data unit (CADU) and correct codeblocks if necessary
- **quad_file** – quadratic table

4.6 MFPS Global Units

There are no major frame processing subsystem (MFPS) global routines.

4.7 PCDS Global Units

In addition to the RSL functions described in Section 3.5, the PCDS includes the following global routines:

- **pcd_1750AToDouble** – converts a MIL-STD-1750A format floating point number to a type double floating point number
- **pcd_ConstructTime** – converts a string representation of spacecraft time in the form YYYY:DDD:HH:MM:SS.xxxxx to a different string in the form YYYYDDDHHMMSSxxxx and vice versa
- **pcd_LagrangeInt** – performs polynomial interpolation using a straightforward implementation of Lagrange's formula
- **pcd_TwosCompConv** – converts a 4-byte two's complement number to double precision

4.8 IDPS Global Units

There are no IDPS global routines.

4.9 LDTS

4.9.1 Global Units

LPS data transfer subsystem (LDTS) global routines are as follows:

- **ldt_AcceptClient** – accepts ECS client socket connection
- **ldt_Broadcast** – sends a message via socket to ECS
- **ldt_CreateServer** – creates receive data delivery notice (DDN) rendezvous server for receiving DDN messages from ECS
- **ldt_GetCurrentEDCTime** – obtains current EDC time and places it into a format for NASA installations
- **ldt_PackHeader** – packs message header with message type and message length information
- **ldt_ProcTermSig** – cleans up process and reports the signal on receipt of a termination signal
- **ldt_ReadSocket** – reads messages from socket
- **ldt_SelectSocket** – checks socket activity using blocking I/O

- **ldt_SetSocketOpts** – sets receive or send socket buffer size and sets client socket to non_blocking I/O
- **ldt_SocketResponse** – responds to status of reading a socket message
- **ldt_TimeDiff** – calculates delta time of two times
- **ldt_UnpackHead** – reads message type and message length from message header
- **ldt_WriteSocket** – writes data to a socket
- **ldt_create_client** – opens client socket connection to a specified host/port
- **ldt_day_to_month** – given the day of year and 4-digit year, returns the month and the day within the month
- **ldt_establish_client** – opens socket connection to ECS server, sends authentication request, and waits for authentication response
- **ldt_free_dan** – frees allocated memory for created data availability notices (DANs)
- **ldt_insert_time_stamp** – builds ISO-8601 time format string
- **ldt_keyscan_ld** – scans keywords from a Parameter Value Language (PVL) buffer
- **ldt_next_string** – obtains address of next string with a protocol message
- **ldt_put_string** – inserts a string to a specified location
- **ldt_save_message** – saves control messages between LPS and ECS into specified location
- **ldt_string_fail** – detects out-of-bound or runaway string

4.9.2 Reuse Code

The LDTS reuses the code from the Data Distribution Facility (DDF) Simulator project, Mission Operations and Systems Development Division (GSFC), in the areas discussed in the following subsections.

4.9.2.1 TCP/IP Sockets

The LDTS reuses the functions to implement Berkeley socket communication. Section 4.9.1 provides the functional description.

- **ldt_AcceptClient**
- **ldt_Broadcast**
- **ldt_CreateServer**
- **ldt_ReadSocket**

- **ldt_SetSocketOpts**
- **ldt_SocketResponse**
- **ldt_WriteSocket**
- **ldt_create_client**
- **ldt_establish_client**

4.9.2.2 Create DAN

The LDTS reuses the following functions to create DAN message:

- **ldt_CreateDAN** – creates DAN message in ASCII format
- **ldt_db_ExtrDANStruct** – extracts LPS L0R file information from database and constructs a DAN message
- **ldt_read_dan_info** – reads a DAN from disk into a buffer
- **ldt_write_dan_info** – writes a DAN from a buffer into a file in ASCII format

4.9.2.3 Miscellaneous

The miscellaneous functions excluding the Transmission Control Protocol/Internet Protocol (TCP/IP) socket functions listed in Section 4.9.1 are also reused from the DDF Simulator project.

Section 5. Portability Issues

5.1 long long

The LPS software makes use of the “long long” type to declare a 64-bit integer. Under ANSI/ISO C on the SGI, this is allowed but results in a warning.

5.2 HPDI Device Driver

See Section 2.3.

5.3 System Software Calls

The following list does not include system library functions called by LPS global functions, Pro*C precompiled DBARs, or the frame synchronization software.

5.3.1 LPS

The LPS globals incorporate the following system library functions:

access	malloc	shmat	strchr
atoi	memcpy	shmctl	strcmp
execv	memmove	shmdt	strcpy
fabs	mkdir	shmget	strlen
fork	msgctl	sigaction	strstr
fprintf	msgget	sigaddset	strtok
getenv	msgrcv	sigemptyset	strtol
gethostname	msgsnd	sprintf	syslog
getpgrp	semctl	sqlglm	time
gettimeofday	semget	stat	
leap	semop	strcat	

5.3.2 MACS

The MACS incorporates the following system library functions:

alarm	getenv	semop	strcat
atof	getpgrp	setpgid	strcmp
atoi	getpid	sigaction	strcpy
closedir	kill	sigdelset	strlen
exit	malloc	sigfillset	strncpy
fclose	opendir	sigismember	strrchr
feof	readdir	signal	strstr
fgets	remove	sigpending	strtok
fopen	rewind	sigprocmask	wait
fprintf	rmdir	sigsuspend	waitpid
free	semctl	sprintf	
fscanf	semget	statfs	

5.3.3 RDCS

The system commands to perform processor isolation and restriction, as well as setting process priority, are SGI specific. These system calls are contained with the rdc_IsolateProcess.c and rdc_DeIsolateProcess.c units.

The system function stacker(1M) is used to communicate with the DLT device. The units that reference this function are rdc_LoadTape.c, rdc_UnloadTape.c, and rdc_TapeBinCount.c.

The system function killall(1M) is used by the rdc_SuspendProcess.c, rdc_ResumeProcess.c, and rdc_Terminate.c units.

The system function stat64(2) is used to obtain file statistics for a raw wideband data file due to the possible size of the file exceeding 32-bit file system limitations. This system function is only referenced by rdc_CaptureCalcAccounting.c.

The unit rdc_vmereset.c was developed by SGI and uses some SGI-specific system commands. This unit should be taken into careful consideration before porting to another platform or operating system.

5.3.4 RDPS

The RDPS incorporates the following system library functions:

open	close
exit	read
memalign	

5.3.5 MFPS

The MFPS incorporates the following system library functions:

open
close
write

5.3.6 PCDS

The PCDS incorporates the following system library functions:

acos	ceil	fmod	pow
asin	cos	isnan	sin
atan	fabs	memcpy	sprintf
atan2	floor	memset	sqrt

5.3.7 IDPS

The IDPS incorporates the following system library functions:

fopen	read	getenv	stat
fclose	write	pipe	getpgrp
fread	fflush	mkfifo	wait
fwrite	fork	unlink	exit

5.3.8 LDTS

The LDTS incorporates the following system library functions:

accept	getenv	read	strchr
access	gethostbyname	realloc	strcmp
bind	getpid	return	strcpy
calloc	gmtime	select	strlen
close	kill	setsockopt	strncmp
connect	leap	sigaction	strtok
exit	listen	sleep	stat
fcntl	malloc	socket	sysinfo
FD_ZERO	memcpy	sprintf	system
FD_SET	memset	sscanf	write
fprintf	nanosleep	strcat	
free	open	strcasecmp	

5.4 Other Portability Issues

The PCDS software makes use of the “#pragma pack(1)” directive. This may not behave the same if it is ported to a different system.

Section 6. Development Environment

6.1 Directory Structure

The LPS directories follow a hierarchical structure. They are under the $LPS_HOME home directory, which is configurable under different platforms. The directories are created by the makedir_lps and makedir_system scripts during system setup. The top level includes the following subdirectories: bin, COTS, data, db, global, journal, man, outfile, rawfile, reports, scripts, tables, tools, troublefile, ui, and seven subsystem directories. Each subsystem directory contains eight subdirectories: bin, data, db, global, include, obj, scripts, and src. Each global or db directory may break down further into the include, obj, and src subdirectories. The src directories under RDCS, IDPS, and LDTS will contain the source files and one or more src**n** (where **n** is a numerical number starting with 1) subdirectories to accommodate multiple programs under an src directory. The purpose of each directory is as follows:

- **bin** – contains the executables
- **COTS** – contains COTS and GOTS software
- **data** – contains data files used in LPS
- **db** – contains database access source files, includes, object codes, and libraries used by the subsystem or LPS
- **global** – contains global routines, includes, object codes, and libraries used by the subsystem or LPS
- **include** – contains include files with .h extension
- **journal** – contains the LPS Journal files
- **man** – contains LPS man pages
- **obj** – contains object codes with the .o extension and source libraries with the .a extension
- **outfile** – contains the LPS L0R output files
- **rawfile** – contains LPS captured raw data files
- **reports** – contains LPS generated reports
- **scripts** – contains scripts used by the subsystem or LPS
- **src** – contains source codes with a .c or .pc extension and Imakefile, lmake, and Makefile
- **tables** – contains tables referenced by LPS
- **tools** – contains software tools used by LPS for development

- **troublefile** – contains the trashed major frame files
- **ui** – contains ORACLE Forms and other user interface files

6.1.1 LPS-Developed Software

LPS-developed software and their locations are as follows:

$LPS_HOME/bin

$LPS_HOME/data

$LPS_HOME/db/include

$LPS_HOME/db/obj

$LPS_HOME/db/src

$LPS_HOME/global/include

$LPS_HOME/global/obj

$LPS_HOME/global/src

$LPS_HOME/man

$LPS_HOME/scripts

$LPS_HOME/tables

$LPS_HOME/tools/bin

$LPS_HOME/tools/include

$LPS_HOME/tools/obj

$LPS_HOME/tools/src

$LPS_HOME/ui/bin

$LPS_HOME/ui/include

$LPS_HOME/ui/obj

$LPS_HOME/ui/src

$LPS_HOME/IDPS//bin

$LPS_HOME/IDPS/data

$LPS_HOME/IDPS/db/include

$LPS_HOME/IDPS/db/obj

$LPS_HOME/IDPS/db/src

$LPS_HOME/IDPS/global/include

$LPS_HOME/IDPS/global/obj

$LPS_HOME/IDPS/global/src

$LPS_HOME/IDPS/include

$LPS_HOME/IDPS/scripts

$LPS_HOME/IDPS/src

$LPS_HOME/LDTS/bin

$LPS_HOME/LDTS/data

$LPS_HOME/LDTS/db/include

$LPS_HOME/LDTS/db/obj

$LPS_HOME/LDTS/db/src

$LPS_HOME/LDTS/global/include

$LPS_HOME/LDTS/global/obj

$LPS_HOME/LDTS/global/src

$LPS_HOME/LDTS/include

$LPS_HOME/LDTS/scripts

$LPS_HOME/LDTS/src

$LPS_HOME/MACS/bin

$LPS_HOME/MACS/data

$LPS_HOME/MACS/db/include

$LPS_HOME/MACS/db/obj

$LPS_HOME/MACS/db/src

$LPS_HOME/MACS/global/include

$LPS_HOME/MACS/global/obj

$LPS_HOME/MACS/global/src

$LPS_HOME/MACS/include

$LPS_HOME/MACS/scripts

$LPS_HOME/MACS/src

$LPS_HOME/MFPS/bin

$LPS_HOME/MFPS/data

$LPS_HOME/MFPS/db/include

$LPS_HOME/MFPS/db/obj

$LPS_HOME/MFPS/db/src

$LPS_HOME/MFPS/global/include

$LPS_HOME/MFPS/global/obj

$LPS_HOME/MFPS/global/src

$LPS_HOME/MFPS/include

$LPS_HOME/MFPS/scripts

$LPS_HOME/MFPS/src

$LPS_HOME/PCDS/bin

$LPS_HOME/PCDS/data

$LPS_HOME/PCDS/db/include

$LPS_HOME/PCDS/db/obj

$LPS_HOME/PCDS/db/src

$LPS_HOME/PCDS/global/include

$LPS_HOME/PCDS/global/obj

$LPS_HOME/PCDS/global/src

$LPS_HOME/PCDS/include

$LPS_HOME/PCDS/scripts

$LPS_HOME/PCDS/src

$LPS_HOME/RDCS/bin

$LPS_HOME/RDCS/data

$LPS_HOME/RDCS/db/include

$LPS_HOME/RDCS/db/obj

$LPS_HOME/RDCS/db/src

$LPS_HOME/RDCS/global/include

$LPS_HOME/RDCS/global/obj

$LPS_HOME/RDCS/global/src

$LPS_HOME/RDCS/include

$LPS_HOME/RDCS/scripts

$LPS_HOME/RDCS/src

$LPS_HOME/RDPS/bin

$LPS_HOME/RDPS/data

$LPS_HOME/RDPS/db/include

$LPS_HOME/RDPS/db/obj

$LPS_HOME/RDPS/db/src

$LPS_HOME/RDPS/global/include

$LPS_HOME/RDPS/global/obj

$LPS_HOME/RDPS/global/src

$LPS_HOME/RDPS/include

$LPS_HOME/RDPS/scripts

$LPS_HOME/RDPS/src

6.1.2 Location of COTS/GOTS Software

COTS/GOTS software contains the ORACLE, HDF, HDF-EOS, RSL, and frame synchronization subdirectories. COTS/GOTS is located in the $COTS_HOME directory, except for ORACLE. The locations of the software are shown in the following subsections.

6.1.2.1 ORACLE

$ORACLE_HOME

6.1.2.2 HDF

$HDF_HOME/bin

$HDF_HOME/include

$HDF_HOME/lib

$HDF_HOME/scripts

$HDF_HOME/src

6.1.2.3 HDF-EOS

$HDF_EOS/lib

6.1.2.4 RSL

$PCDS_GLOBAL_INC

$PCDS_GLOBAL_OBJ

$PCDS_GLOBAL_SRC

6.1.2.5 Frame Synchronization

$FS_HOME/include

$FS_HOME/obj

$FS_HOME/scripts

$FS_HOME/src

6.2 Environment Files

The LPS uses two C Shell scripts (*.lpsrc* and *.lpsdevrc*) to initialize the environment variables. *.lpsrc* sets up the LPS operational/test environment for running LPS. *.lpsdevrc* sets up the LPS development environment for compiling and building LPS.

To execute these two scripts, the following lines should be added to a user's .cshrc file:

unsetenv PATH MANPATH

setenv LPS_HOME <the LPS root directory>

source $LPS_HOME/.lpsrc

(or source $LPS_HOME/.lpsdevrc)

NOTE: *.lpsdevrc* also calls *.lpsrc* so it is not necessary to source both scripts explicitly.

6.2.1 .lpsrc

The following environment variables and their values in *.lpsrc* are set as follows:

setenv PATH /usr/local/bin:/usr/bin:/bin:/usr/etc:/etc:/usr/bsd:/usr/sbin:/sbin

setenv MANPATH /usr/share/catman:/usr/share/man:/usr/catman:/usr/local/man

setenv LD_LIBRARY_PATH /usr/lib

setenv LD_LIBRARYN32_PATH /usr/lib32

setenv LD_LIBRARY64_PATH /usr/lib64

setenv X11HOME /usr/bin/X11

setenv PATH $X11HOME/:$PATH

setenv ORACLE_HOME /usr/app/oracle/product/7.3.2

setenv PATH $ORACLE_HOME/bin:$PATH

setenv LD_LIBRARY_PATH $ORACLE_HOME/lib:$LD_LIBRARY_PATH

setenv ORACLE_PATH $PATH

setenv ORACLE_SID LPS

setenv ORACLE_TERM xterm

setenv TNS_ADMIN /usr/app/oracle/client/developer2000/1.3.1/tns

setenv TWO_TASK lps

setenv LPS_BIN $LPS_HOME/bin

setenv PATH $LPS_BIN/:$PATH

setenv MANPATH $LPS_HOME/man:$MANPATH

setenv LPS_DANFILE_PATH $LPS_HOME/DAN

setenv LPS_DDNFILE_PATH $LPS_HOME/DDN

setenv LPS_JOURNAL_PATH /u03/tmp

setenv LPS_OUTFILE_PATH $LPS_HOME/outfile

setenv LPS_RAWFILE_PATH $LPS_HOME/rawfile

setenv LPS_REPORT_PATH $LPS_HOME/reports

setenv LPS_TAPE_DEV /dev/rmt/tps131d5

setenv LPS_TABLE_PATH $LPS_HOME/tables

setenv LPS_TEMPFILE_PATH /u03/tmp

setenv LPS_TROUBLEFILE_PATH $LPS_HOME/troublefile

setenv LPS_IAS_PARMS_PATH $LPS_HOME/iasparms

setenv LPS_CONT_SCHED_PATH $LPS_HOME/schedules

setenv LPS_PRINTER_DEVICE /dev/plp

setenv LPS_TAPE_LIBRARY_DEV /dev/scsi/sc131d510

setenv RDC_DEVICE /dev/hpdiB

setenv RDC_STATUS_INTERVAL 30

setenv RDC_THRESH_SYSTEMDISK 0.01

setenv LPS_CAPTURE_PROCESSOR 1

6.2.2 .lpsdevrc

The following environment variables and their values in *.lpsdevrc* are set as follows:

setenv PURIFYHOME /usr/pure/purify

setenv PATH $PURIFYHOME/:$PATH

setenv MANPATH $PURIFYHOME/man:$MANPATH

setenv HDF_HOME $LPS_HOME/COTS/hdf/4.0r2_IRIX_5.3

setenv HDF_BIN $HDF_HOME/bin

setenv HDF_INC $HDF_HOME/include

setenv HDF_OBJ $HDF_HOME/lib

setenv HDF_SRC $HDF_HOME/src

setenv HDF_SCRIPTS $HDF_HOME/scripts

setenv HDF_EOS $LPS_HOME/COTS/hdf/hdfeos

setenv PATH $LPS_HOME/tools/bin:$PATH

setenv PATH $LPS_HOME/IDPS/bin:$PATH

setenv PATH $LPS_HOME/LDTS/bin:$PATH

setenv PATH $LPS_HOME/MACS/bin:$PATH

setenv PATH $LPS_HOME/MFPS/bin:$PATH

setenv PATH $LPS_HOME/PCDS/bin:$PATH

setenv PATH $LPS_HOME/RDCS/bin:$PATH

setenv PATH $LPS_HOME/RDPS/bin:$PATH

setenv PATH $LPS_HOME/ui/bin:$PATH

setenv CVINSTRLIB $HOME

NOTE: *.lpsdevrc* sources $LPS_HOME/.env and $LPS_HOME/COTS/hdf/hdfeos/bin/sgi/hdfeos_env.csh sets up additional environment variables.

6.2.3 Possible Upgrade Problems

Some root environment variables, such as LPS_HOME, ORACLE_HOME, HDF_HOME, and HDF_EOS, need to be updated when a new version of each product is installed on the machine.

6.3 Compiler Considerations

6.3.1 Configuration Flags

Because LPS software is required to run under ANSI mode instead of an SGI platform's default extended ANSI mode, the extended ANSI (-xansi) flag is replaced by an -ansi flag in /usr/lib/X11/config/sgi.cf. The following line shows the change in /usr/lib/X11/config/sgi.cf:

```
#define sgiCCOptions -ansi -D_BSD_SIGNALS sgiABIopts sgiABIdefs
```

NOTE: The RDCS uses the original SGI platform configuration file, sgi.cf with the -xansi flag set.

6.3.2 Expected Warning Messages

6.3.2.1 General

The following warning messages are produced when compiling the LPS source containing long long type to produce the object under ANSI mode:

cfe: Warning 799: pcd_db_GetFirstWrsScene.c, line 1236: ‘long long’ is not standard ANSI. (3.1.1)

typedef long long __uint64_t;

----------^

This keyword/type is not defined in strict ANSI mode.

cfe: Warning 799: /u03/LPS/b3/global/include/lps_annotated_cadu.h, line 78: ‘long long’ is not standard ANSI. (3.1.1)

typedef long long

----------^

This keyword/type is not defined in strict ANSI mode.

The following warning messages are produced when linking the LPS source objects or libraries under SGI IRIX 6.2 operating system:

ld: WARNING 84: /u02/home/l7xlsrv/inteam/si/b3.1/global/obj/libglobal.a is not used for resolving any symbol.

ld: WARNING 84: /usr/app/oracle/product/7.3.2/lib/libxa.a is not used for resolving any symbol.

ld: WARNING 84: /usr/app/oracle/product/7.3.2/lib/libsqlnet.a is not used for resolving any symbol.

ld: WARNING 84: /usr/app/oracle/product/7.3.2/lib/libncr.a is not used for resolving any symbol.

ld: WARNING 84: /usr/app/oracle/product/7.3.2/lib/libsqlnet.a is not used for resolving any symbol.

ld: WARNING 84: /usr/app/oracle/product/7.3.2/lib/libclient.a is not used for resolving any symbol.

ld: WARNING 84: /usr/app/oracle/product/7.3.2/lib/libgeneric.a is not used for resolving any symbol.

ld: WARNING 84: /usr/app/oracle/product/7.3.2/lib/libnlsrtl3.a is not used for resolving any symbol.

ld: WARNING 84: /u02/home/l7xlsrv/inteam/si/b3.1/COTS/hdf/hdfeos/lib/sgi/libhdfeos.a is not used for resolving any symbol.

ld: WARNING 84: /u02/home/l7xlsrv/inteam/si/b3.1/COTS/hdf/hdfeos/lib/sgi/libGctp.a is not used for resolving any symbol.

ld: WARNING 84: /u02/home/l7xlsrv/inteam/si/b3.1/COTS/hdf/4.0r2_IRIX_5.3/lib/libmfhdf.a is not used for resolving any symbol.

ld: WARNING 84: /u02/home/l7xlsrv/inteam/si/b3.1/COTS/hdf/4.0r2_IRIX_5.3/lib/libdf.a is not used for resolving any symbol.

ld: WARNING 84: /u02/home/l7xlsrv/inteam/si/b3.1/COTS/hdf/4.0r2_IRIX_5.3/lib/libjpeg.a is not used for resolving any symbol.

ld: WARNING 84: /u02/home/l7xlsrv/inteam/si/b3.1/COTS/hdf/4.0r2_IRIX_5.3/lib/libz.a is not used for resolving any symbol.

ld: WARNING 84: /u02/home/l7xlsrv/inteam/si/b3.1/COTS/frame_synchronization/obj/libfs.a is not used for resolving any symbol.

ld: WARNING 84: /usr/lib/libsocket.so is not used for resolving any symbol.

ld: WARNING 84: /usr/lib/libnsl.so is not used for resolving any symbol.

ld: WARNING 85: definition of _ffs in /usr/lib/libnsl.so preempts that definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of ffs in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of gethostname in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of clnt_create in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 85: definition of _clnt_create_vers in /usr/lib/libnsl.so preempts that definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of clnt_create_vers in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of clnt_sperror in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of clnt_perror in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of clnt_sperrno in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of clnt_perrno in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 85: definition of _clnt_perrno in /usr/lib/libnsl.so preempts that definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of clnt_spcreateerror in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of clnt_pcreateerror in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 85: definition of _clntraw_create in /usr/lib/libnsl.so preempts that definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of clntraw_create in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 85: definition of _callrpc in /usr/lib/libnsl.so preempts that definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of callrpc in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of clnttcp_create in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of clntudp_bufcreate in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of clntudp_create in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of netname2user in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 85: definition of _netname2user in /usr/lib/libnsl.so preempts that definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of netname2host in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 85: definition of _netname2host in /usr/lib/libnsl.so preempts that definition in /usr/lib/libc.so.

ld: WARNING 85: definition of _getnetname in /usr/lib/libnsl.so preempts that definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of getnetname in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of user2netname in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of host2netname in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of pmap_set in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of pmap_unset in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of pmap_getmaps in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: WARNING 85: definition of _pmap_getmaps in /usr/lib/libnsl.so preempts that definition in /usr/lib/libc.so.

ld: WARNING 134: weak definition of pmap_getport in /usr/lib/libnsl.so preempts that weak definition in /usr/lib/libc.so.

ld: Giving up after printing 50 warnings. Use -wall to print all warnings.

6.3.2.2 RDCS

The rdc_DeviceFunctions.c unit cannot be compiled with the -ansi flag. This unit contains software that communicates with the HPDI device. For compatibility with the SGI-developed device driver, the extended ANSI flag (-xansi) must be used when compiling this unit.

6.4 Imakefiles

Due to the nature of configurability and portability of Imake, LPS chooses it to create the Makefiles for building LPS-developed software instead of using the traditional Makefiles. Although the Imake provides easier, simpler, open-ended mechanism to generate the Makefiles on various platforms, two files need to be configured to cooperate platform-specific information: Imake.rules and Imake.tmpl

The following lines in /usr/lib/X11/config/Imake.rules are added to the Imake rule base to compile LPS source files with extension .c or .pc into the object codes, to clean up files under the source directory, and to generate executables:

```
/*
 * Pro C compiler – Running pc unit through Pro C compile to get c units.
 */
#ifndef TurnPCUnitsIntoCUnits
#define TurnPCUnitsIntoCUnits() rm -f $*.c; cp $*.pc $*.c;                @@\
    chmod u+w $*.c;                                                       @@\
    $(CC) -P -D_NO_LONGLONG $(INCLUDES) $(DEFINES) $*.c; \                @@\
    sed “/#ident/d” $*.i > $*.ipc; \                                      @@\
    $(ORACLE_PROC) iname=$*.ipc $(PROFLAGS); \                            @@\
    rm $*.ipc $*.i
```

```
#endif /* TurnPCUnitsIntoCUnits */
/*
 * Pro C compiler – generate rules to build necessary things during make all.
 */
#ifndef NormalProCTarget
#define NormalProCTarget()                                  @@\
.SUFFIXES: .pc .c .o .ln                                    @@\
                                                            @@\
.pc.c:                                                      @@\
TurnPCUnitsIntoCUnits()                                     @@\
                                                            @@\
FORCE: $(SRCS)
#endif /* NormalProCTarget */
/*
 * CleanTarget2 – generate rules to remove any garbage files
 */
#ifndef CleanTarget2
#define CleanTarget2()                                      @@\
clean::                                                     @@\
        $(RM_CMD) FilesToClean2                             @@\
                                                            @@\
cleaner:clean                                               @@\
        $(RM_CMD) FilesToCleaner2 ExtraFilesToClean "#"*    @@\
        $(RM_CMD) `ListPCUnitsAsCUnits()`                   @@\
                                                            @@\
ProofCleanTarget()
#endif /* CleanTarget2 */
/*
 * DependTargetWithPC – generate rules to compute dependencies for all files listed
```

```
 * in $(SRCS).
 */
#ifndef DependTargetWithPC
#define DependTargetWithPC()                                    @@\
DependDependency()                                              @@\
depend::                                                        @@\
    RunProgram(DEPEND,$(DEPENDFLAGS)        --      $(ALLDEFINES)
    $(DEPEND_DEFINES) -- $
(SOURCESC) $(SOURCESPC))
@@\
#endif /* DependTargetWithPC */
/*
 * ObjectCompileWithPC – compile fragment for a normal object file
 */
#ifndef ObjectCompileWithPC
#define ObjectCompileWithPC(options) -@if [ -f $*.pc ]; then \      @@\
    TurnPCUnitsIntoCUnits(); \                                  @@\
    fi                                                          @@\
    RemoveFile($@)                                              @@\
    $(CC) -c $(CFLAGS) options $*.c
#endif
/*
 * NormalLibObjCompileWithPC – compile fragment for a normal library object file
 */
#ifndef NormalLibObjCompileWithPC
#define NormalLibObjCompileWithPC(options) ObjectCompileWithPC(options)
#endif
/*
 * NormalLibraryObjectRulWithPC – for simple libraries
 */
```

```
#ifndef NormalLibraryObjectRuleWithPC
#define NormalLibraryObjectRuleWithPC()                                 @@\
.c.Osuf:                                                                @@\
    NormalLibObjCompileWithPC($(_NOOP_))
#endif /* NormalLibraryObjectRuleWithPC */
/*
 * NormalLibraryObjectRulWithPC – for simple libraries
 */
#ifndef ListPCUnitsAsCUnits
#define ListPCUnitsAsCUnits() ls *.pc 2>/dev/null | sed "s/\.pc/\.c/p" | uniq |
tr '\012' '\040'
#endif /* ListPCUnitsAsCUnits */
/*
 * ProjectLinkRule – link a program
 */
#ifndef ProjectLinkRule
#define ProjectLinkRule(program,options,objects,libraries) \
$(CCENVSETUP) $(PURIFY) $(QUANTIFY) $(CCLINK) -o program options objects
    libraries $(EXTRA_LOAD_FLAGS)
#endif /* ProjectLinkRule */
/*
 * NormalProgramTarget – generate rules to compile and link the indicated
 * program; since it does not use any default object files, it may be used for
 * multiple programs in the same Imakefile.
 */
#ifndef ProjectProgramTarget
#define ProjectProgramTarget(program,objects,deplibs,locallibs,syslibs)     @@\
ProgramTargetName(program): objects deplibs                             @@\
    RemoveTargetProgram($@)                                             @@\
    ProjectLinkRule($@,$(LDOPTIONS),objects,locallibs syslibs)          @@\
```

```
                                                                    @@\
clean::                                                             @@\
    RemoveFile(ProgramTargetName(program))
#endif /* NormalProgramTarget */
```

The following lines in /usr/lib/X11/config/Imake.tmpl are modified as shown:

```
#ifndef FilesToClean
#define FilesToClean *.CKP *.ln *.BAK *.bak *.i *.ipc tmpa* *.Osuf core errs ,*
*~ *.a .emacs_* tags TAGS make.log MakeOut PureFilesToClean QuantifyFilesToClean
#endif
#ifndef FilesToClean2
#define FilesToClean2 *.CKP *.BAK *.bak *.i *.ipc tmpa* *.Osuf core errs ,* *~ *
.a .emacs_* tags TAGS make.log MakeOut *.lis *.err *.met
#endif
#ifndef FilesToCleaner2
#define FilesToCleaner2 *.ln *.o *.d *.tcov *.a PureFilesToClean QuantifyFilesToClean
#endif
```

6.4.1 General Considerations

To generate the Makefiles using Imake, each source directory must have an Imakefile exist and the filename must be called “Imakefile” exactly. The top portion of Imakefile displays the options when calling the lmake script to compile the source files. The ORACLE directories and ORACLE precompile information usually have to be modified according to the version of ORACLE database installed in the system. The LPS global libraries may also vary if the COTS/GOTS software has been changed.

Depending on the purpose of the Imakefile, the bottom portion of Imakefile varies: one for generating libraries and one for generating executable targets. The examples of these Imakefile templates are as follows:

Imakefile for generating libraries:

```
/**************************************************************************
#
# Make Usage
#
#**************************************************************************/
```

```
help:
	@ echo “Usage:”
	@ echo “ lmake help”
	@ echo “ to see this help message.”
	@ echo “”
	@ echo “ lmake [ debug | nodebug | cclint | optimize ]”
	@ echo “ to create $(LIBRARY).”
	@ echo “”
	@ echo “lmake lint”
	@ echo “ to run lint on *.c and create llib-ldbp.ln.”
	@ echo “”
	@ echo “ lmake cadre”
	@ echo “ to run reverse engineering bmdl tool on *.c files”
	@ echo “”
	@ echo “ lmake clean”
	@ echo “ to remove *%, *~, *.lis, *.i, core, and dbp files.”
	@ echo “”
	@ echo “ lmake cleaner”
	@ echo “ to remove *%, *~, *.lis, *.i, core, dbp, *.o, *.ln, precompiled units,”
	@ echo “ $(LIBRARY), *.d , and *.tcov.”
/*****************************************************************************
#
# Extract Objects and Sources
#
#*****************************************************************************/
#define PassCDebugFlags ‘CDEBUGFLAGS=$(CDEBUGFLAGS)’
OBJS= $(SRCS:.c=.o) XCOMM all object files
SOURCESCPC= $(SOURCESPC:.pc=.c) XCOMM all pc units turned into c files
/*****************************************************************************
```

```
*
* ORACLE directories
*
*********************************************************************/
ORACLE_LIB = $(ORACLE_HOME)/lib
ORACLE_PROC = $(ORACLE_HOME)/bin/proc
ORACLE_PRELIB= $(ORACLE_HOME)/precomp/lib
LD_LIBRARY_PATH = $(ORACLE_HOME)/lib

LIBRPC = $(ORACLE_LIB)/libncr.a
LIBSQLNET = $(ORACLE_LIB)/libsqlnet.a
NETLIBD = $(LIBSQLNET) $(LIBRPC) $(LIBSQLNET)

LIBGENERIC = $(ORACLE_LIB)/libgeneric.a
LIBCOMMON = $(ORACLE_LIB)/libcommon.a
LIBCLIENT = $(ORACLE_LIB)/libclient.a
LIBORA = $(LIBCLIENT) $(LIBCOMMON) $(LIBGENERIC)

LIBEPC = $(ORACLE_LIB)/libepc.a

LIBCORE = $(ORACLE_LIB)/libcore3.a
LIBCV6 = $(ORACLE_LIB)/libc3v6.a
LIBNLSRTL = $(ORACLE_LIB)/libnlsrtl3.a
CORELIBD = $(LIBNLSRTL) $(LIBCV6) $(LIBCORE) $(LIBNLSRTL) $(LIBCORE)
    $(LIBNLSRTL)

TTLIBD = $(NETLIBD) $(LIBORA) $(LIBSQLNET) $(LIBRPC) $(LIBSQLNET)
    $(LIBORA) $(CORELIBD) $(LIBEPC)

LIBSQL = $(ORACLE_LIB)/libsql.a
```

```
LIBXA = $(ORACLE_LIB)/libxa.a
PROLDLIBS = $(LIBXA) $(LIBSQL) $(TTLIBD)

LIBMOD = $(ORACLE_PRELIB)/libmod.a
LIBPCC = $(ORACLE_PRELIB)/libpcc.a
LIBPGP = $(ORACLE_PRELIB)/libpgp.a
LIBPROC2 = $(ORACLE_PRELIB)/libproc2.a
EXTRALIB = $(LIBMOD) $(LIBPCC) $(LIBPGP) $(LIBPROC2)
ORACLELIBS = $(PROLDLIBS) $(EXTRALIB)
/*************************************************************************
*
* ORACLE precompile information
*
*************************************************************************/
PROFLAGS=ireclen=160 oreclen=160 select_error=no mode=ANSI code=ANSI_C
    DBMS=V7
PCCFLAGS=include=$(PCCINC) ireclen=160 oreclen=160 select_error=no
/*************************************************************************
#
# LPS Global libraries
#
#************************************************************************/
XGLOBAL_LIB = $(GLOBAL_OBJ)/libglobal.a
XDB_LIB = $(DB_OBJ)/libdb.a
HDF_LIB = $(HDF_EOS)/lib/sgi/libhdfeos.a \
$(HDF_EOS)/lib/sgi/libGctp.a \
$(HDF_HOME)/lib/libmfhdf.a \
$(HDF_HOME)/lib/libdf.a \
$(HDF_HOME)/lib/libjpeg.a \
$(HDF_HOME)/lib/libz.a
```

```
FS_LIB = $(FS_OBJ)/libfs.a

GLOBALLIBS = $(XGLOBAL_LIB) $(XDB_LIB) $(XGLOBAL_LIB) $(PROLDLIBS)
    $(HDF_LIB) $(FS_LIB)

/*************************************************************************

#

# LPS Global includes

#

#************************************************************************/

HDF_INC = -I$(HDF_EOS)/include \

    -I$(HDF_HOME)/include

GLOBALINCS = -I$(GLOBAL_INC) -I$(DB_INC) -
    I$(ORACLE_HOME)/precomp/public $(HDF_INC)

/*************************************************************************

#

# LPS Global dependencies

#

#************************************************************************/

GLOBALDEPS = $(GLOBALLIBS)

/*************************************************************************

#

# Local libraries

#

#************************************************************************/

GLOBAL_LIB = $(XXXX_GLOBAL_OBJ)/libXXXXglobal.a

DB_LIB = $(XXXX_DB_OBJ)/libXXXXdb.a

LOCALLIBS = $(GLOBAL_LIB) $(DB_LIB) $(GLOBAL_LIB)

/*************************************************************************

#

# Local includes

#
```

```
#*****************************************************************************/
LOCALINCS = -I$(XXXX_GLOBAL_INC) -I$(XXXX_DB_INC) -I$(XXXX_INC)

/*****************************************************************************
#
# LPS Global dependencies
#
#*****************************************************************************/
LOCALDEPS = $(LOCALLIBS)
/*****************************************************************************
#
# Build dependencies
#
#*****************************************************************************/
LOCAL_LDFLAGS= $(ORACLE_LDFLAGS)
LOCAL_LIBRARIES = $(LOCALLIBS) $(GLOBALLIBS)
DEPLIBS = $(LOCALDEPS) $(GLOBALDEPS)
INCLUDES = $(LOCALINCS) $(GLOBALINCS)
SYS_LIBRARIES = -lm -lsocket -lnsl
DEFINES = -DIRIX_VERSION $(EXTRA_DEFINES)
OBJDIR = $(XXXX_GLOBAL_OBJ)
/*****************************************************************************
#
# Create Library
#
#*****************************************************************************/
LIBRARY = XXXXglobal
debug nodebug cclint optimize : lib$(LIBRARY).a
NormalProCTarget()
```

```
NormalLibraryObjectRuleWithPC()
NormalLibraryTarget($(LIBRARY),$(OBJS))
InstallLibrary($(LIBRARY),$(OBJDIR))

DependTargetWithPC()
LintTarget()
```

Imakefile for generating executables/programs:

```
/*****************************************************************************
#
# Make Usage
#
#*****************************************************************************/
help:
	@ echo "Usage:"
	@ echo " lmake help"
	@ echo " to see this help message."
	@ echo ""
	@ echo " lmake [ debug | nodebug | cclint | optimize ]"
	@ echo " to create $(LIBRARY)."
	@ echo ""
	@ echo " lmake lint"
	@ echo " to run lint on *.c and create llib-ldbp.ln."
	@ echo ""
	@ echo " lmake cadre"
	@ echo " to run reverse engineering bmdl tool on *.c files"
	@ echo ""
	@ echo " lmake clean"
	@ echo " to remove *%, *~, *.lis, *.i, core, and dbp files."
```

```
	@ echo “”
	@ echo “ lmake cleaner”
	@ echo “ to remove *%, *~, *.lis, *.i, core, dbp, *.o, *.ln, precompiled units,”
	@ echo “ $(LIBRARY), *.d , and *.tcov.”
/*************************************************************************
#
# Extract Objects and Sources
#
#*************************************************************************/
#define PassCDebugFlags ‘CDEBUGFLAGS=$(CDEBUGFLAGS)’
OBJS= $(SRCS:.c=.o) XCOMM all object files
SOURCESCPC= $(SOURCESPC:.pc=.c) XCOMM all pc units turned into c files
/*************************************************************************
*
* ORACLE directories
*
*************************************************************************/
ORACLE_LIB = $(ORACLE_HOME)/lib
ORACLE_PROC = $(ORACLE_HOME)/bin/proc
ORACLE_PRELIB= $(ORACLE_HOME)/precomp/lib
LD_LIBRARY_PATH = $(ORACLE_HOME)/lib

LIBRPC = $(ORACLE_LIB)/libncr.a
LIBSQLNET = $(ORACLE_LIB)/libsqlnet.a
NETLIBD = $(LIBSQLNET) $(LIBRPC) $(LIBSQLNET)

LIBGENERIC = $(ORACLE_LIB)/libgeneric.a
LIBCOMMON = $(ORACLE_LIB)/libcommon.a
LIBCLIENT = $(ORACLE_LIB)/libclient.a
```

```
LIBORA = $(LIBCLIENT) $(LIBCOMMON) $(LIBGENERIC)


LIBEPC = $(ORACLE_LIB)/libepc.a


LIBCORE = $(ORACLE_LIB)/libcore3.a

LIBCV6 = $(ORACLE_LIB)/libc3v6.a

LIBNLSRTL = $(ORACLE_LIB)/libnlsrtl3.a

CORELIBD = $(LIBNLSRTL) $(LIBCV6) $(LIBCORE) $(LIBNLSRTL) $(LIBCORE)
    $(LIBNLSRTL)


TTLIBD = $(NETLIBD) $(LIBORA) $(LIBSQLNET) $(LIBRPC) $(LIBSQLNET)
    $(LIBORA) $(CORELIBD) $(LIBEPC)


LIBSQL = $(ORACLE_LIB)/libsql.a

LIBXA = $(ORACLE_LIB)/libxa.a

PROLDLIBS = $(LIBXA) $(LIBSQL) $(TTLIBD)


LIBMOD = $(ORACLE_PRELIB)/libmod.a

LIBPCC = $(ORACLE_PRELIB)/libpcc.a

LIBPGP = $(ORACLE_PRELIB)/libpgp.a

LIBPROC2 = $(ORACLE_PRELIB)/libproc2.a

EXTRALIB = $(LIBMOD) $(LIBPCC) $(LIBPGP) $(LIBPROC2)

ORACLELIBS = $(PROLDLIBS) $(EXTRALIB)

/*************************************************************************

*

* ORACLE precompile information

*

*************************************************************************/

PROFLAGS=ireclen=160 oreclen=160 select_error=no mode=ANSI code=ANSI_C
    DBMS=V7

PCCFLAGS=include=$(PCCINC) ireclen=160 oreclen=160 select_error=no
```

```
/*****************************************************************************
#
# LPS Global libraries
#
#*****************************************************************************/
XGLOBAL_LIB = $(GLOBAL_OBJ)/libglobal.a
XDB_LIB = $(DB_OBJ)/libdb.a
HDF_LIB = $(HDF_EOS)/lib/sgi/libhdfeos.a \
$(HDF_EOS)/lib/sgi/libGctp.a \
$(HDF_HOME)/lib/libmfhdf.a \
$(HDF_HOME)/lib/libdf.a \
$(HDF_HOME)/lib/libjpeg.a \
$(HDF_HOME)/lib/libz.a
FS_LIB = $(FS_OBJ)/libfs.a
GLOBALLIBS = $(XGLOBAL_LIB) $(XDB_LIB) $(XGLOBAL_LIB) $(PROLDLIBS)
    $(HDF_LIB) $(FS_LIB)
/*****************************************************************************
#
# LPS Global includes
#
#*****************************************************************************/
HDF_INC = -I$(HDF_EOS)/include \
    -I$(HDF_HOME)/include
GLOBALINCS = -I$(GLOBAL_INC) -I$(DB_INC) -
    I$(ORACLE_HOME)/precomp/public $(HDF_INC)
/*****************************************************************************
#
# LPS Global dependencies
#
#*****************************************************************************/
```

```
GLOBALDEPS = $(GLOBALLIBS)
/*************************************************************************
#
# Local libraries
#
#*************************************************************************/
GLOBAL_LIB = $(XXXX_GLOBAL_OBJ)/libXXXXglobal.a
DB_LIB = $(XXXX_DB_OBJ)/libXXXXdb.a
LOCALLIBS = $(GLOBAL_LIB) $(DB_LIB) $(GLOBAL_LIB)
/*************************************************************************
#
# Local includes
#
#*************************************************************************/
LOCALINCS = -I$(XXXX_GLOBAL_INC) -I$(XXXX_DB_INC) -I$(XXXX_INC)

/*************************************************************************
#
# LPS Global dependencies
#
#*************************************************************************/
LOCALDEPS = $(LOCALLIBS)
/*************************************************************************
#
# Build dependencies
#
#*************************************************************************/
LOCAL_LDFLAGS= $(ORACLE_LDFLAGS)
LOCAL_LIBRARIES = $(LOCALLIBS) $(GLOBALLIBS)
```

```
DEPLIBS = $(LOCALDEPS) $(GLOBALDEPS)
INCLUDES = $(LOCALINCS) $(GLOBALINCS)
SYS_LIBRARIES = -lm -lsocket -lnsl
DEFINES = -DIRIX_VERSION $(EXTRA_DEFINES)
BINDIR = $(XXXX_BIN)
/************************************************************************
*
* Build Major Subsystem Target
*
************************************************************************/
TARGET = rcvddn
debug nodebug cclint optimize purify purifynodebug purifyoptimize : $(TARGET)
NormalProCTarget()
```

To use the Imakefile templates, simply plug in the subsystem identifier into XXXX and specify the LIBRARY or TARGET names.

6.4.1.1 lmake

Once a Makefile is generated under each source directory using Imake via Imakefile, lmake provides a convenient way to run different options of make. Therefore, each source directory must have a lmake script exist. A example of lmake script is as follows:

```
case $1 in
debug) EXTRA_DEFINES='-DDEBUG'
    export EXTRA_DEFINES
    make -f Makefile $1 "CDEBUGFLAGS=-ansi -fullwarn -g"
    ;;
nodebug) make -f Makefile $1 "CDEBUGFLAGS=-ansi -fullwarn"
    ;;
cclint) make -f Makefile $1 "CDEBUGFLAGS=-ansi -fullwarn -wlint"
    ;;
optimize) make -f Makefile $1 "CDEBUGFLAGS=-ansi -fullwarn -O2"
    ;;
```

```
gprof) make -f Makefile $1 “CDEBUGFLAGS=-ansi -fullwarn -xpg -O2”
    ;;
tcov) make -f Makefile $1 “CDEBUGFLAGS=-ansi -fullwarn -xa”
    ;;
purify) EXTRA_DEFINES=‘-DDEBUG’
    export EXTRA_DEFINES
    PURIFY=‘purify -first-only -chain-length=20 -suppression-file-names=“.pu
rify, .purify.irix”‘
    export PURIFY
    make -f Makefile $1 “CDEBUGFLAGS=-ansi -fullwarn -g”
    ;;
purifynodebug) PURIFY=‘purify -first-only -chain-length=20 -suppression-file-nam
es=“.purify, .purify.irix”‘
    export PURIFY
    make -f Makefile $1 “CDEBUGFLAGS=-ansi -fullwarn”
    ;;
purifyoptimize) PURIFY=‘purify -first-only -chain-length=20 -suppression-file-na
mes=“.purify, .purify.irix”‘
    export PURIFY
    make -f Makefile $1 “CDEBUGFLAGS=-ansi -fullwarn -O2”
    ;;
lint) make -f Makefile $1
    ;;
clean) make -f Makefile $1
    ;;
cleaner) make -f Makefile $1
    ;;
install) make $1
    ;;
```

```
help) make -f Makefile $1
    ;;
Makefile) make -f Makefile $1
    ;;
depend) make $1 "CDEBUGFLAGS="
    ;;
*) echo "*** Invalid make option ***"
    make -f Makefile help
    ;;
esac
```

6.4.1.2 Imakefile Usage

If the lmake file does not exist in current source directory, the following procedures are recommended to generate the Makefile and compile the source files:

>xmkmf	---------->	create Makefile
>make clean	---------->	remove old files
>make depend	---------->	search dependencies
>make	---------->	create libraries or executables
>make install	---------->	install libraries or executables into destination
>make lint	---------->	create lint information

If the lmake script file exists in the source directory, simply run lmake and all the usages and options will be displayed on the screen. However, if the Makefile does not exist under the source directory in the beginning, the xmkmf command must be run to generate a Makefile before all of the lmake options can be exercised.

6.4.2 ORACLE

6.4.2.1 Pro*C Precompiler

An ORACLE Pro*C precompiler is a programming tool that allows the developer to embed Structured Query Language (SQL) or PL/SQL statements in a high-level source program (including C).The precompiler accepts the source program written in C as input, translates the embedded SQL statements into standard ORACLE runtime library calls, and generates a modified source program that the developer can compile, link, and execute using a C compiler in the usual way.

To run the Pro*C precompiler to precompile the DBARs, the proc command with the following options is used:

proc IRECLEN=160 ORECLEN=160 SELECT_ERROR=NO MODE=ANSI CODE=ANSI_C DBMS=V7

IRECLEN=160 : Specifies the record length of the input file be 160

ORECLEN=160 : Specifies the record length of the output file be 160

SELECT_ERROR=NO : No error is generated when a single row select returns too many rows than the host can accommodate.

MODE=ANSI : The program compiles fully with the ANSI SQL standard.

CODE=ANSI_C : Specifies to generate full function prototypes, which confirm to the ANSI C standard, by the PRO*C precompiler.

DBMS=V7 : Specifies the ORACLE to follow the semantic and syntactic rules of ORACLE7.

6.4.2.2 Forms 4.5 Files

Table 6–1 shows the file extensions for each type of Forms 4.5 module and storage format.

Table 6–1. File Extensions for Forms 4.5

Module	Binary (Design)	Executable Runfile
Form	.fmb	.fmx
Menu	.mmb	.mmx

The form/menu source file is created using the Forms graphical user interface (GUI) designer executable, f45desm. It creates the binary file with the extension .fmb/.mmb. The executable runfile (.fmx/.mmx) is created either using the designer or using the command line command.

To generate a form/menu executable from the Designer, select the File->Administration menu ->Generate of the Object Navigator.

To generate a form executable from the command line, the following command is used:

f45genm module=form_name.fmb userid/passwd

To generate a menu executable from the command line, the following command is used:

f45genm module=menu_name userid/passwd module_type=menu

6.4.2.3 Reports 2.5 Files

Table 6–2 explains the ORACLE report file types:

Table 6–2. ORACLE Report Files

File Type	Contents	Format
.rdf	Single report definition	Binary executable, can be modified by using Designer
.rep	Single report, does not contain comments or source code	Binary executable, cannot be modified

The report binary executable file is created using the Forms GUI designer executable, r25desm. It creates the binary file with the .rdf extension. The slimmer executable runfile (.rep) is created either using the designer or using the command line command.

To generate a .rep file from the Report Designer, select the File->Generate menu of the Report Designer.

To generate a .rep file from the command line, the following command is used:

r25convm userid/passwd source=report_name stype=rdffile dest=target_report_file_name dtype=repfile overwrite=yes batch=yes

6.4.3 Possible Upgrade Problems

6.4.3.1 Operating System Dependencies

The LPS may run into problems when the operating system is upgraded to the new version. The current operating system in use is SGI IRIX 6.2. Pay special attention to those units that contain the nonportable codes when upgrading the operating system.

6.4.3.2 ORACLE DBMS Dependencies

The LPS database may run into problems when the ORACLE DBMS is upgraded. As discussed in Section 6.4.1, the ORACLE directories and precompile information in the Imakefile may be necessary to change. The LPS DBMS currently uses ORACLE 7.3.2.

6.4.3.3 HDF/HDF-EOS Dependencies

Because the software of HDF/HDF-EOS depends on the running operating system, there may be a problem using it if the operating system is upgraded. The LPS currently uses HDF 4.0r2.

Section 7. Testing LPS Software

7.1 Simulators

7.1.1 ECS Simulator

To test the control messages between the LDTS and ECS, an ECS_simulator is built based on the Interface Control Document (ICD) Between ECS and the Landsat 7 System. It is enhanced from DDF Simulator, Mission Operations and Systems Development Division (GSFC). The ECS_simulator contains 10 simulations available, but only the ninth and tenth are of interest. Because the simulator is GOTS software, contact GSFC for support if software changes are needed.

ECS_simulator consists of two programs:

1. **ecs_sim** – the actual simulator that talks to the remote host
2. **ecs_ui** – the user interface that communicates with ecs_sim

The programs run in separate windows on an X terminal. User commands are passed from ecs_ui to ecs_sim in a command file. Socket message files (binary or text) are passed between the programs and are kept for later analysis/use by the user.

Before making any software change to ECS_simulator, the following references are required reading:

- ICD Between ECS and the Landsat 7 System
- README.users_guides under ECS source directories
- README.deliv_ltr, included in delivery with source code
- make_ecs (the make file for both programs), included in delivery with source code (The prolog provides information on the directory configuration and is required for building the executables.)

7.1.2 Major Frame Processing and Image Data Processing Simulator for Testing Payload Correction Data

PCDS testing is much simplified if a simulated environment is used. The simulated environment runs faster than the full mac_startl0r and is more easily controlled. Several custom test tools have been developed for PCDS testing. These tools include programs for simulating the PCD program's runtime environment and for manipulating test data. The test tools are unsupported and indifferently documented. They are located in the tools directory.

7.1.2.1 mac – simulates top-level processing functions of mac_startl0r

This program creates the requisite shared-memory segments and the pcd-to-idp FIFO and invokes the mfp, pcd, and idp programs (all are assumed to reside in the current working directory), waits for them to complete, and writes their return status to stdout.

mac is invoked as follows:

example% mac file-name contact-sequence-ID file-version-number

where *file-name* is the name of a file containing unpacked PCD cycles, *contact-sequence-ID* is the contact sequence identifier associated with the contact in the LPS database, and *file-version-number* is any valid file version number. The latter two arguments are passed to the pcd program.

CAUTION

mac sometimes hangs indefinitely whenever a child process exits with a failure exit code. In this case, killing mac with a SIGINT will shut the process down, but shared memories and FIFOs will not have been deallocated. In this case, use the ipc_cleanup utility to deallocate shared resources.

The following source files are compiled to produce the mac executable.

- **mac_Main** – main function
- **mac_processControl** – functions to start and wait for child processes to exit
- **mac_processControl.h** – contains prototypes for mac_processControl functions and #include's lps_types.h

7.1.2.2 mfp – simulates the mfp program

mfp reads a file of unpacked PCD and transmits it to pcd in the correct format. mfp sends appropriate end-of-subinterval and end-of-contact messages at the end of the run. When invoked, mfp main issues a prompt and waits for user input before continuing. This allows attachment of a debugger to the pcd program before it receives any input.

WARNING

A bug causes mfp to reprompt forever unless the first character typed is "y."

mfp's output is determined by the value of variables and constants set at compile time in the following way. The scenes variable is set to the number of scenes to be output. For each scene, mfp generates several major frames equal to MAJOR_FRAMES_PER_SCENE (a static constant int variable). For each major frame, mfp reads 4 * MAX_VCDUS_PER_ETM_MAJ_FR (another static constant int variable) bytes from the PCD input file. mfp exits with a fatal error if the input file contains fewer than the requisite bytes. mfp does not use any PCD in the input file beyond the requisite bytes.

Several versions of this program simulate different interesting test conditions:

- **mfp.old** – sends simulated Enhanced Thematic Mapper (ETM+) major frames to the idp simulator as well as PCD to the pcd program

- **mfp.normal** – sends only PCD to the pcd program with an end-of-contact message to the idp simulator
- **mfp.skip20** – discards bytes representing the contents of every twentieth CADU to simulate missing virtual channel data units (VCDUs), and sets the missing VCDU count field in the lpsPCDInfoStruct structure passed to pcd
- **mfp.2si** – sends two subintervals to pcd

NOTE

To use one or the other of these executables, the appropriate executable should be copied to a file named "mfp."

The following source files are compiled to produce the mfp executable:

- **mfp_Main** – main function for all versions of mfp
- **mfp_Main.nowait** – alternative version of the main function that does not wait for pcd to start up
- **mfp_datagen.h** – prototypes for data generating functions; this file is used by all versions of mfp
- **mfp_datagen.normal** – data generating functions for mfp.normal
- **mfp_datagen.c.original** – data generating functions for mfp.old
- **mfp_datagen.skip20** – data generating functions for mfp.skip20
- **mfp_datagen.2si** – data generating functions for mfp.2si

7.1.2.3 idp – simulates the idp program

idp reads the pcd-to-idp FIFO and writes the contents of each message to stdout. There are two versions of idp:

- **idp** – does not expect shared-memory segments to be provided by the mfp simulator; use with all versions of mfp except mfp.old
- **idp.old** – reads both shared-memory segments from the mfp simulator and the pcd FIFO (NOTE: The program does not read the pcd FIFO until the end of the subinterval and thereafter does not read the mfp shared-memory interface until all messages from pcd for the subinterval have been received.)

The following source files are compiled to produce idp executables:

- **idp_Main.c** – compile to produce idp
- **idp_Main.c.old** – compile to produce idp.old

7.1.2.4 Other Tools

Several other unsupported but useful test tools are provided for PCDS testing:

- **bitflip** – C program (source is bitflip.c) that reads a specified file of unpacked PCD and writes a file containing the contents of the input file with 1 bit flip introduced into each PCD word triplet. The bit flip occurs in bit 0 of the first word in the first triplet and then proceeds bitwise through each word. Invocation method is "bitflip *file*." bitflip creates the output file "*file*.errors" in the current working directory.

- **bskip** – C program (source is bskip.c) that copies a binary file, skipping a specified number of bytes at the beginning and replacing them with the same number binary zeros at the end. bskip is useful for creating truncated versions of files of unpacked PCD that can still be handled by mfp. The invocation method is "bskip *file number-of-bytes*." bskip creates the output file "*file*.bskip" in the current working directory.

- **bzero** – C program (source is bzero.c) that creates a file containing a specified number of binary zero bytes. The invocation method is "bzero *file number-of-0s*." bzero creates the output file "*file*" in the current working directory.

- **create_test_tables.sql** – SQL script that creates the set of LPS database tables used by the pcd program without constraints. This script is useful when testing in a shared environment because it provides private versions of shared database tables. The script also allows the developer to remove database constraints from tables when the constraints are impeding debugging. Invoke the script in SQL*Plus by typing "start create_test_tables."

- **drop.sql** – SQL script that drops the set of tables created by create_test_tables.sql. Invoke the script in SQL*Plus by typing "start drop."

- **dtots** – C program (source is dtots.c) that accepts a real number interpreted as the time in seconds since January 1, 1993, and produces a string representing the time. The real number time format is used extensively in pcd, and this utility is useful when attempting to interpret a time in that format. The invocation method is "dtots *time-as-real-number*." The string is written to stdout.

- **ephem** – C program (source is ephem.c) that extracts and displays attitude and ephemeris values from a file of packed PCD. To produce a file of packed PCD from a file of unpacked PCD, type the following.

  example% fmtpcd *unpacked-file-name* | awk '{print $2}' > *packed-file-name*

  See the fmtpcd description for more information.

  CAUTION

  ephem outputs incorrect latitude and longitude values for each attitude and ephemeris. Ignore them.

- **fmtpcd** – C program (source is fmtpcd.c) that reads a specified file and outputs a formatted hex dump of the unpacked PCD cycles in the file. Each cycle appears on a separate line. The invocation method is "fmtpcd *file*." The hex dump is written to stdout.

- **getpcd** – C program (source is getpcd.c) that extracts unpacked PCD 4-tuples from a raw wideband data set and writes them to a file. getpcd only works if the first synchronization pattern begins at bit 0 of the first byte and there are no bit slips. The invocation method is "getpcd *file.*" The unpacked PCD is written to a file named *file*.pcd in the current working directory.

- **gha** – C program (source is gha.c) that computes a Greenwich hour angle (GHA) given a UT1 time. The invocation method is "gha *YYYY DDD HH:MM:SS.xxxxx*." The GHA is written to stdout.

- **ipc_cleanup** – shell script that deletes all shared-memory segments, FIFOs, and semaphores allocated to the invoker's user ID

- **mag** – C program (source is mag.c) that computes a vector magnitude. The invocation method is "mag *X Y Z*." The magnitude is written to stdout.

- **mjd** – C program (source is mjd.c) that computes a modified Julian day value given a year and Julian day. The invocation method is "mjd *YYYY DDD*." The modified Julian day is written to stdout.

- **tstod** – C program (source is tstod.c) that computes a time represented as number of seconds since January 1, 1993. The invocation method is "tstod *YYYY DDD HH:MM:SS.xxxxx*" The computed time is written to stdout.

7.2 Viewing Shared Memory

7.2.1 RDPS-to-MFPS

Two concurrent sessions of CaseVisionDebugger were used to view the shared memory between the RDPS and the MFPS: one session for the MFPS and the other for the RDPS.

7.2.2 MFPS-to-PCDS

A PCDS simulator was used that logged the main items of data going into the shared memory in a file. For a detailed view of shared memory between the PCDS and the MFPS, two concurrent sessions of CaseVisionDebugger were used: one session for the MFPS and the other for the PCDS.

7.2.3 MFPS-to-IDPS

An IDPS simulator was used that logged the main items of data going into the shared memory in a file. For a detailed view of shared memory between the IDPS and the MFPS, two concurrent sessions of CaseVisionDebugger were used: one session for the MFPS and the other for the IDPS.

7.3 COTS and GOTS

7.3.1 ORACLE

7.3.1.1 Referential Integrity Constraint Considerations

ORACLE supports the use of foreign key integrity constraints to enforce referential data integrity. The inter-relationship between the LPS database tables (attributes) is as follows:

- Parent tables
  - RDC_ACCT (CONTACT_SEQUENCE_ID)
  - PROCESSING_VERSION_INFO (CONTACT_SEQUENCE_ID, FILE_VERSION_NUMBER)
  - SUB_INTV (CONTACT_SEQUENCE_ID, FILE_VERSION_NUMBER, SUB_INTV_SEQUENCE_ID)

  Among these parent tables, PROCESSING_VERSION_INFO (CONTACT_SEQUENCE_ID) references RDC_ACCT (CONTACT_SEQUENCE_ID). SUB_INTV (CONTACT_SEQUENCE_ID, FILE_VERSION_NUMBER) references PROCESSING_VERSION_INFO (CONTACT_SEQUENCE_ID, FILE_VERSION_NUMBER).

- Dependent tables (attributes) referencing SUB_INTV (SUB_INTV_SEQUENCE_ID)
  - BANDS_PRESENT (SUB_INTV_SEQUENCE_ID)
  - BAND_GAIN_STATES (SUB_INTV_SEQUENCE_ID)
  - IDP_ACCT (SUB_INTV_SEQUENCE_ID)
  - LDT_FILE_GROUP_INFO (SUB_INTV_SEQUENCE_ID)
  - LPS_FILE_INFO (SUB_INTV_SEQUENCE_ID)
  - MFP_ACCT (SUB_INTV_SEQUENCE_ID)
  - MFP_MJF_ACCT (SUB_INTV_SEQUENCE_ID)
  - PCD_ACCT (SUB_INTV_SEQUENCE_ID)
  - PCD_MJF_ACCT (SUB_INTV_SEQUENCE_ID)
  - PCD_SCENE_ACCT (SUB_INTV_SEQUENCE_ID)

- Dependent tables (attributes) referencing PROCESSING_VERSION_INFO (CONTACT_SEQUENCE_ID, FILE_VERSION_NUMBER)
  - LDT_DAN_INFO (CONTACT_SEQUENCE_ID, FILE_VERSION_NUMBER)
  - LDT_FILE_SET_INFO (CONTACT_SEQUENCE_ID, FILE_VERSION_NUMBER)

– RDP_ACCT (CONTACT_SEQUENCE_ID, FILE_VERSION_NUMBER)

When running a test by interactively entering records into the tables using SQL*Plus, make sure that the child table record's foreign key value exists as a referenced key value in the parent table record.

The foreign key constraints defined in child tables in the LPS database are set up with the "on delete cascade" option. It allows deletion of referenced key values in the parent table that have dependent records and causes ORACLE to automatically delete dependent records from the child table to maintain the referential integrity.

For example,

- SUV_INTV table has a record with SUB_INTV_SEQUENCE_ID = 10.
- MFP_ACCT table has a record with SUB_INTV_SEQUENCE_ID = 10.
- PCD_ACCT table has a record with SUB_INTV_SEQUENCE_ID = 10.

The "SQL> delete from SUV_INTV where SUB_INTV_SEQUENCE_ID = 10" command will delete records with SUB_INTV_SEQUENCE_ID = 10 from all three tables.

7.3.1.2 Other Integrity Constraints

In addition to the referential integrity constraint, three more constraint types are defined to maintain the LPS data integrity: Not Null, Primary Key, and Check. The constraints naming convention is as follows:

- pk_FullTableName – primary key constraint
- fk_FullTableName – foreign key constraint
- ck_TableNameAcronym_AttributeNameAbbreviation – check constraint
- nn_TableNameAcronym_AttributeNameAbbreviation – not null constraint

For example, the BANDS_PRESENT table with three attributes has the following constraints. All key constraints are indicated at the SUB_INTV_SEQUENCE_ID attribute:

BANDS_PRESENT (SUB_INTV_SEQUENCE_ID, PCD_CYCLE_TIME, BAND_PRESENT)

pk_bands_present for primary key constraint

fk_bands_present for foreign key constraint referencing SUB_INTV(SUB_INTV_SEQUENCE_ID)

ck_bp_sub_intv_seq_id for check constraint

check (0 < SUB_INTV_SEQUENCE_ID)

nn_bp_sub_intv_seq_id for not null constraint

While running the LPS processes, if a constraint violation condition occurs, the developer can find out its full information by retrieving it from the data dictionary ALL_CONSTRAINTS table keying on the constraint name under SQL*Plus.

For example, if the error message logged in the LPS Journal is "ORA-02290: check constraint (APPLDBA.CK_BP_SUB_INTV_SEQ_ID) violated.", the following command interactively entered using SQL*Plus will return all the information about the check constraint violation.

```
SQL> select * from ALL_CONSTRAINTS where constraint_name =
'CK_BP_SUB_INTV_SEQ_ID';
```

7.3.1.3 Others

To run a test, it is sometimes necessary to manipulate data interactively using SQL*Plus before or during running a test process. In such case, make sure that the changes be committed for the data changes to be accessible by the test process by issuing the "SQL> commit;" command.

7.3.2 EOSView

EOSView is an interactive X Window-based tool developed by the ECS project for viewing HDF-EOS files. It differs from other HDF compatible tools in that it can interpret the complex HDF-EOS specific structures such as swaths and geolocation tables. It is capable of displaying data in numeric form and rendering images. Data is displayed in scrollable windows.

7.3.3 vshow

vshow is a command line-based utility provided by NCSA for dumping the contents of HDF files in ASCII format. This utility is not sophisticated, but it works. It cannot interpret the HDF-EOS structures; however, it does a fair job of displaying their content based on the lower level HDF structures from which they are built. It does not render images. vshow is useful for examining data with which EOSView has difficulty, such as characters. NCSA provides it along with the HDF library.

7.3.4 LinkWinds

LinkWinds is a sophisticated GUI-based tool developed at the Jet Propulsion Laboratory (JPL) for viewing and evaluating scientific data and that can read HDF files as input. It cannot interpret HDF-EOS structures, but is able to read and display the image data contained in an LPS band file. This tool incorporates many advanced data analysis features and is useful to a user who wishes to not only view the contents of an LPS swath, but also to study it. LinkWinds is available on several UNIX-based platforms. The SGI/IRIX version works well, but will only run with SGI workstations because it uses SGI-specific firmware. It does not work with a generic X terminal. LinkWinds is available free of charge from JPL. Visit their Web site (http://linkwinds.jpl.nasa.gov/) for details.

7.3.5 gtedit

gtedit is a Generic Telemetry Simulator utility used to edit (review/modify) the telemetry data. It is invoked using the following command:

```
gtedit [-h] [-p path] [-s | -v | -b blocksize] filename
```

where

filename = data file to edit

-b blocksize = length in bytes of fixed length data units in the data file

- p path = alternate path for a temporary file that is used by the gtedit (In absence of this flag, /tmp is used for the purpose.)

-v = data file contains variable length data units

-s = data file contains data unit summaries

-h = Display help screen

For example, the gtedit -b 1040 /LPS/b3/mfps/data/97-123-10:12:14.cpt command is used to edit the /LPS/b3/mfps/data/97-123-10:12:14.cpt data file, which contains data units of 1040 bytes each.

Once gtedit displays the data file, one block of data at a time, edit commands are used to edit the data file. Most edit commands are similar to the vi editor commands. The z command is used to toggle between hex and binary (bits) display. The i command is used to enter edit mode to edit either a hex digit (a nibble) or a bit. <Esc> is used to exit edit mode. /xx is used to search for hex byte xx. :w and :q are used to save the data file and to quit gtedit, respectively. -h is used to show other edit commands.

Section 8. Design Decisions

8.1 General

8.1.1 Interface Consideration

8.1.1.1 FIFOs

The LPS project uses the UNIX Message Queues IPC facilities to provide low bandwidth, controlled communication among it subsystems.

When LPS is brought up, the MACS invokes lps_RsrcAlloc() to start creating IPC resources. It creates an LPS shared-memory resource control shared-memory segment and an LPS FIFO control shared-memory segment. These two shared-memory segments are used to manage the LPS shared-memory resources and the LPS FIFO resources. They also contain the resource access information for LPS subsystems. When the control shared-memory segments have been created, lps_RsrcAlloc() invokes lps_RsrcAllocFIFO() to create LPS FIFO queues.

When L0R data processing is completed or an error is encountered at L0R data processing, the MACS invokes lps_RsrcDealloc() to remove the LPS FIFO queues. This will guarantee that no unwanted FIFO queues will be left in the system.

Subsystems or processes requiring access to the LPS FIFO queues call lps_FIFOOpen() to attach to the LPS FIFO queues created by the MACS. The subsystems or processes can then call lps_FIFOSend() and lps_FIFOReceive() to exchange information. In the meantime, the subsystems or processes can specify whether or not to wait when reading or writing from/to the LPS FIFO queues. After data processing is done or an error encountered, the subsystems or processes call lps_FIFOClose() to detach themselves from LPS FIFO queues (Figure 8–1).

8.1.1.2 Shared Memory

LPS project uses the UNIX System V Shared Memory IPC facilities to provide high-bandwidth, high-volume controlled communication among it subsystems. The semaphore is developed on top of the shared memory to control the uses of shared memory.

As described in Section 8.1.1.1, LPS shared-memory segments are created by the MACS at the start of LPS L0R processing. When LPS is brought up, the MACS invokes lps_RsrcAlloc() to start creating IPC resources. It creates an LPS shared-memory resource control shared-memory segment and an LPS FIFO control shared-memory segment. These two shared-memory segments are used to manage the LPS shared-memory resources and the LPS FIFO resources. They also contain the resource access information for LPS subsystems. When the control shared-memory segments have been created, lps_RsrcAlloc() invokes lps_RsrcAllocShm() to create LPS shared-memory resources. The MACS invokes lps_RsrcDealloc() to remove the LPS shared memories from the system at the completion of L0R data processing or an error is encountered at L0R data processing.

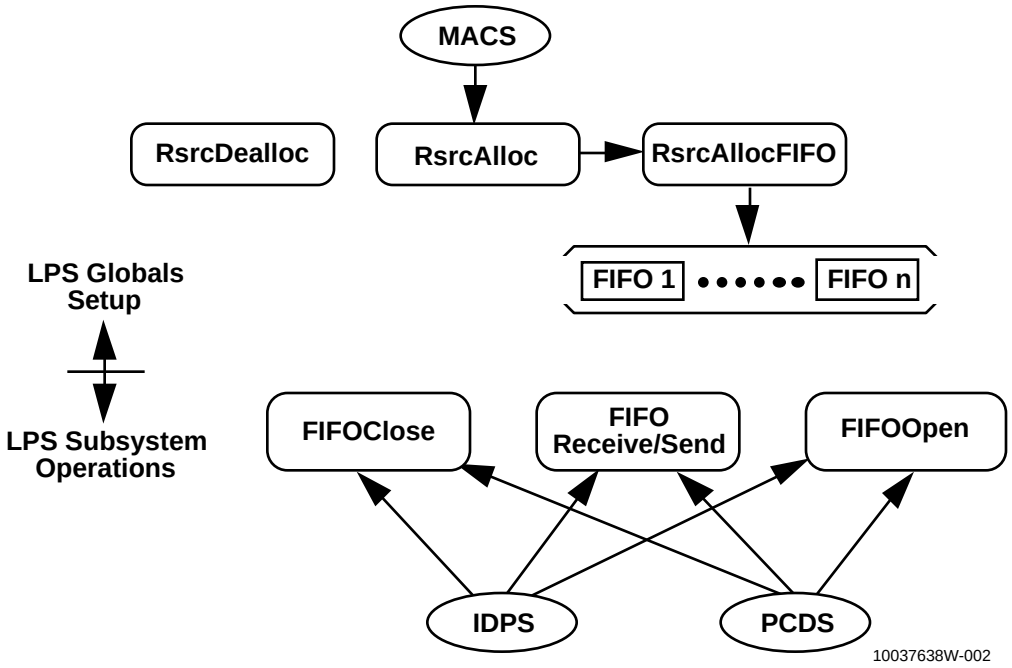

Figure 8–1. LPS FIFO Operations

All LPS IPC resources will be assigned unique keys that are constant and are defined in the global libraries. The MACS uses the keys to get the effective keys, which are then used to create or remove LPS IPC resources. The other subsystems or processes provide the assigned keys to the global libraries, and the global libraries map out the effective keys for the subsystems to gain access to LPS IPC resources (Figure 8–2).

Once the shared resource control shared-memory segment is created, lps_RsrcAllocShm() invokes lps_ShmCreate() to create the shared-memory segments used in LPS. Based on the specific number of shared memories used by the subsystems, the number of blocks in each shared memory, and the block size needed by the subsystems, lps_ShmCreate() creates the shared-memory segments. After the shared-memory segments are successfully created, a semaphore set is created accordingly for each shared-memory segment and will be used to control and synchronize the access of shared-memory blocks. The created shared-memory segments are then divided into a number of fixed-size blocks and the block addresses, and associated shared-memory segment and information are stored in the shared resource control information table. Initially, all the shared-memory blocks are initialized as free (write) blocks in the block pool. The subsystem calls lps_ShmGetWrBlk() to retrieve a free block for writing and calls lps_ShmPutWrBlk() to put the written block into the active (read) block pool after it finishes the write block.

Likewise, when a subsystem needs to retrieve information from the active (read) block pool, it calls lps_ShmGetRdBlk() to obtain a read block and calls lps_ShmPutRdBlk() to return the read block back into the free block pool. Low-level global library functions (lps_ShmAddListTail(), lps_ShmRemListTail(), and lps_ShmRemListHead) are used to insert and remove the shared-memory blocks in the pools. The blocks in the free and active pools are managed in a first-in-last-out (FILO) pattern. On top of the FILO control mechanism, the blocks are also guarded by

the semaphores to prevent the racing condition among the subsystems. The semaphores are also used to provide the subsystems with the option of whether or not to wait for a block to become available.

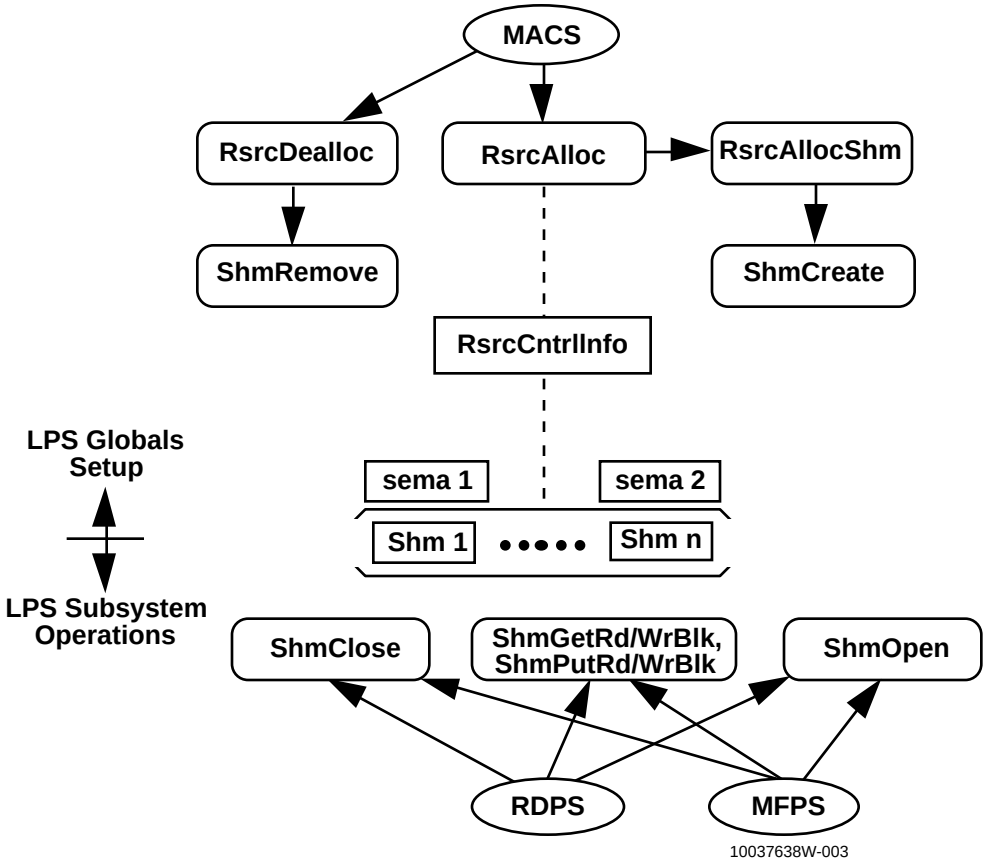

Figure 8–2. LPS Shared-Memory Operations

8.1.1.3 Information Passed Via the Database

Type of Information Passed

The LPS database management system (DBMS) is used to store LPS status and accounting information, image data Q&A information, and metadata information. In addition, the LPS application uses database tables for IPC mechanisms among subsystems to reduce the coupling of LPS subsystems.

Database Commits

Each subsystem process connects (lps_db_Connect) to the LPS database to retrieve, insert, delete, and update data. When it disconnects [lps_db_Disconnect (LPS_COMMIT) or lps_db_Disconnect (LPS_NOCOMMIT)] from the database, it should pass COMMIT as an argument to lps_db_Disconnect to end the current transaction and to make permanent all changes performed in the transaction.

When it is necessary to make an intermediary commit without disconnecting from the database, the lps_db_Commit function can be used.

8.1.2 Rollback with L0R Failure

When the L0R processing of data of a contact period fails, all the records generated and inserted into the LPS database during the L0R processing are to be rolled back (mac_db_RollbackL0R).

ORACLE supports the use of foreign key integrity constraints to enforce referential data integrity. The foreign key constraints defined in child tables in the LPS database are all set up with the "on delete cascade" option. Therefore, deleting the record in the Processing_Version_Info parent table triggers automatic deletion of dependent records in the child tables that reference the deleted record in the parent table.

The Process_Id table is an independent table that does not depend on any other table. The L0R processing indicator record in the Process_Id table is deleted explicitly by mac_db_RollbackL0R.

8.1.3 Error-Handling Philosophy

The LPS project relies on UNIX system facilities to handle the system errors and log error messages.

8.1.3.1 System Signal Handling

The LPS project uses the Berkeley Software Distribution (BSD) UNIX signal facilities to catch the error signal generated by the system when it encounters any system violations. LPS subsystems call the lps_ProcessInit global routine to set up signal handling.

8.1.3.2 Error Logging

The LPS project uses the basic UNIX system services to provide status and error reporting. LPS subsystems call the lps_LogMessage()global routine to log the status or error messages to the LPS Journal error log file using the UNIX syslog function. This function also provides the option to log the messages to the standard error file for debugging purposes.

8.1.4 Database Access Routines

DBARs retrieve, insert, delete, and update records in the database during L0R processing. There are LPS systemwide global database access routines prefixed with lps_db_ and subsystem-level database access routines prefixed with xxx_db_, where xxx denotes a subsystem (RDCS, RDPS, MFPS, PCDS, IDPS, LDTS, or MACS).

DBARs are written using the ORACLE7 Pro*C by embedding SQL or PL/SQL in application programs written in C.

8.2 Management and Control Subsystem

The MACS is responsible for the system-level control and monitoring of LPS devices and processes. It provides the interface medium between LPS and the operator. MACS main functions include

- Generating LPS metadata files and associated accounting files
- Starting and stopping manual or automatic data capture
- Starting and stopping L0R processing
- Starting and stopping archival to tape
- Ingesting contact schedules from the Mission Operations Center (MOC)
- Ingesting calibration parameter files from the Image Assessment System (IAS)

The MACS consists of 12 processes that all operate independently. Each process is started by the operator from the GUI from a command line at the UNIX shell prompt or by another process. To keep each process simple, the LPS database is used as a means of IPC. All access to the database is through DBARs. This separates the functional processing and the database data processing.

8.3 Raw Data Capture Subsystem

8.3.1 rdc_Capture

Mizar Versus HPDI

The biggest issue with rdc_Capture, and rdc_Transmit, is the underlying receive and transmit hardware. Many of the software design decisions were driven by the choice of hardware and the device driver implementation.

The initial hardware device for capture was a Mizar implementation. This implementation provided LPS with a COTS board that provided for data receipt at 75 Mbps, but required a custom serial-to-parallel interface card, and could only transmit data at approximately 30 Mbps.

In mid-1996, an HPDI hardware device was investigated for data capture and transmission. Its advantages were that the board included a COTS serial-to-parallel interface and it transmitted data at above 75 Mbps. This change in the hardware device, however, had ramifications on the software.

Due to Mizar limitations, the device could not be preempted during data receipt. In fact, if the device was preempted, the system had a tendency to panic and reboot. Therefore, rdc_Capture was initially designed and coded without the use of UNIX signals to handle process shutdown. Instead, an ASCII file was used to communicate shutdown directives.

When the capture device switched to the HPDI implementation, the preempting limitation no longer existed. This allowed a move to UNIX signal communications, which had several advantages:

1. Because the ASCII file no longer existed, the capture process and its communication mechanism were self contained. This avoided the need for special code to handle disk space limitations and file I/O issues.
2. SIGALRM could be used to timeout the process at scheduled stop, rather than having to poll the system time. This eliminated a spawned RDCS executable that performed the polling, which saved some performance.

3. Preparing a signal handler for termination communication allowed use of an LPS global function (lps_ProcessInit) for process initialization. This allowed the RDCS capture and transmit functions to initialize identically to the other LPS processes.

4. Preparing the signal handler for termination also provided for software handling of additional problems such as segmentation violations, bus errors, etc.

The device driver for the Mizar and HPDI devices were written by SGI. The device driver interface software was written by LPS team members. To keep these functions isolated from the rest of the RDCS capture and transmit software, they were all placed into a single C module (rdc_DeviceFunctions.c) for ease of software maintenance.

A final limitation existed within the HPDI device itself. The device did not work properly if the Versa-Module European (VME) board was not reset prior to a data capture or transmit. Due to this limitation, source was obtained from SGI to perform this reset. The resulting executable is called rdc_vmereset. This executable is forked by rdc_Capture and rdc_Transmit prior to enabling the device. It is important to note that SGI does not support this VME reset software.

Input

The only input to the rdc_Capture process (other than the raw wideband data) is the process (command line) arguments. The format of the process arguments are, for the most part, legacy formats. In fact, several have been removed or changed since LPS Build 1.

- **-l Capture-Source** – This option used to be named LGS-Channel-ID, which explains the ‘l’ for the option. The name was changed because the capture source was not always going to be the Landsat Ground Station (LGS). Unfortunately, the option was not changed because the letter ‘c’ was taken by other LPS processes to mean Contact-ID, so the ‘l’ remains.

  In addition, Capture-Source used to be a required argument. Because it was decided that terminating the process because a capture source was not provided seemed unnecessary, “NULL” capture sources were allowed. This change affected the LPS global database access routine lps_db_GetRDCInfo.

- **-i** – This option specifies that the process is to isolate and restrict a processor and run with the highest possible nondegrading priority. This feature was designed into the rdc_Capture from the software requirements phase. How the contention of system resources would affect the performance of the data capture was not known. At this point, it is fairly sure that this option will not be required, but, nonetheless, it is still available.

  It is important to note that the rdc_Capture executable must be owned by “root,” and must have the Set Group ID set for this option to succeed. For example, from the UNIX prompt, “ls -l” would result in

  ```
  -r-sr-sr-x   1   root   group   ######   May 31 12:00   rdc_Capture
  ```

  Note the “s” in the permissions.

- **-s** – This option suspends all currently running L0R processes. This feature was designed into rdc_Capture for the same reasons as the -i option. At this point, it is fairly sure that this option will not be required, but, nonetheless, it is still available.

  It is important to note that the mechanism for accomplishing this suspension is through the SIGSTOP/SIGCONT UNIX signals. A couple of caveats exist for this to work. First, the process that initiates L0R processing must be called mac_startl0r, and this process must initialize itself as a process group leader [see setpgid(2)]. Second, the rdc_Capture executable must be owned by "root," and must have the Set Group ID set for this option to succeed. See the -i option for an example.

- **-b Start-Date-&-Time** – This option currently has no functional effect on the RDCS Capture process. It is simply used to place into the RDC_ACCT database table so that the LPS maintains a record of the scheduled start time for the data capture contact period. This option is, probably, only going to be used by the MACS Automatic Capture process to associate the scheduled start time with the value from the SCHEDULES database table.

  The format of the *Start-Date-&-Time* argument was defined during the software requirements phase and has never changed.

- **-e Stop-Date-&-Time** – This option has two purposes. First, similar to the -b option, it serves as a link to the SCHEDULES database table to provide a record of the scheduled stop time. Second, and more importantly, it is used to calculate the duration of the data capture. If the *Stop-Date-&-Time* argument cannot be used to calculate the duration, a default value of 840 seconds is used. 840 is the worst-case duration (0' to 0') of a Landsat 7 contact period (14 minutes). If a longer duration is needed, a valid *Stop-Date-&-Time* will supersede the default value.

  The format of *Stop-Date-&-Time* was defined during the software requirements phase and never changed. A duration in seconds was considered, but never implemented. However, a duration in seconds would probably be more intuitive.

Output

The RDCS primary output is the raw wideband data file. The name is currently specified as YY-DDD-HH:MM:SS_String-ID.data, where YY-DDD-HH:MM:SS is the actual data capture start time and String-ID is the host name on which data capture is executed. This filename definition was defined in such a manner that it would always be unique. Even the case is covered where a string went down and the file was moved to another string for L0R processing.

rdc_Capture has a derived requirement to be able to run unimpeded with or without access to the LPS database. This requirement drove the creation of an additional output file called the RDCS Accounting file. The name is currently specified as YY-DDD-HH:MM:SS_String-ID.acct, where YY-DDD-HH:MM:SS is the actual data capture start time and String-ID is the host name on which data capture is executed.

The RDCS Accounting file is an ASCII file containing, essentially, mirrored information to that contained within the RDC_ACCT database table. Because the capture software is the sole owner

of the RDC_ACCT database table, the capture software had to have a method for retaining the relevant fields if the database was not available.

Notes

The original raw wideband filename had a .cpt extension. This extension did not adequately describe the file, so it was changed to .data. This change should not have had any effect on the LPS software because the filename format was never defined as an interface. L0R processing was intended to use as input whatever filename was placed into the RDC_ACCT database table. Unfortunately, miscommunication occurred, and the LPS global lps_ValidateRDCOutfileName was written to validate the RDCS filename prior to processing raw wideband data to L0R. This unit should have never been written and should eventually be removed. This unit has since been modified to ignore the filename extension as a workaround.

8.3.2 rdc_DeleteFiles

rdc_DeleteFiles is simply an encapsulation of the rdc_db_DeleteRDCFiles RDCS global unit. This process encapsulates the function to allow the raw wideband data files to be deleted from the command line. All raw wideband data deletion is handled by rdc_db_DeleteRDCFiles. This was done because the raw wideband data file deletion had to be performed by the rdc_Save process, and had to be forked/executed by the MACS.

Input

The only input to rdc_DeleteFiles is the process (command line) arguments. The format of the process arguments have been slightly modified since LPS Build 1, but are essentially unchanged.

- **-f filename** – This option specifies the complete path and filename of the raw wideband data file to be deleted. This filename is used as the database key into the RDC_ACCT table to make appropriate table updates pending data deletion. In addition, this filename is used to identify the raw wideband data file, as well as the raw wideband data accounting file requiring deletion.

- **-u** – This option is optional. It specifies an unconditional delete of the raw wideband data files. If provided, rdc_DeleteFiles will not check the L0R processing state of the raw wideband data file.

Output

On successful deletion, this process should have deleted the raw wideband data files (if L0R processed or -u option provided) and updated the archival and online state of the raw wideband data.

Notes

rdc_DeleteFiles currently does not call rdc_Init to perform initialization. If rdc_Init were used, this process would be more consistent with the other RDCS processes and unnecessary source code could be removed, e.g., the call to lps_db_Connect and getopt(1) loop.

rdc_DeleteFiles should add more robust command line argument validation. Currently, the only check made is argc < 2 produces an error. Therefore, the rdc_DeleteFiles -u call is allowed, despite requiring the -f filename option.

rdc_DeleteFiles should add a call to rdc_LogShutdownMessage(LPS_SUCCESS) prior to exiting with a successful status. In addition, this process should replace the call to LPS_LOGMESSAGE with a call to rdc_LogShutdownMessage(LPS_FAILURE) prior to exiting with a failure status.

8.3.3 rdc_GenLabel

rdc_GenLabel simply generates a tape label associated with the specified raw wideband data file. This process is a simple encapsulation of, essentially, two functions: rdc_db_LoadLabelParms and rdc_PrintLabel. rdc_GenLabel was set up as a standalone process that could be forked/executed by other processes and executed directly from the command line.

Input

The only input to rdc_GenLabel is the process (command line) arguments. The format of the process arguments have been slightly modified since LPS Build 1, but are essentially unchanged.

- **-f filename** – This option specifies the complete path and filename of the raw wideband data file to be associated with the tape label. It is used as the database key into the RDC_ACCT table to extract the necessary tape label parameters.

Output

On successful completion of rdc_GenLabel, a tape label associated with the provided raw wideband data file is sent to the printer device defined by the LPS_PRINTER_DEVICE environment variable.

Notes

rdc_GenLabel currently does not call rdc_Init to perform initialization. If rdc_Init were used, this process would be more consistent with the other RDCS processes, and unnecessary source code could be removed, e.g., the call to lps_ProcessInit, the call to rdc_GetArgs, and extraction of the current environment variables.

rdc_GenLabel should add a call to rdc_LogShutdownMessage(LPS_SUCCESS) prior to exiting with a successful status. In addition, this process should replace the calls to LPS_LOGMESSAGE and rdc_ShutDown with a call to rdc_LogShutdownMessage(LPS_FAILURE) prior to exiting with a failure status.

rdc_GenLabel should add a call to rdc_Init to extract environment variables to be used during processing. As a result, rdc_PrintLabel should be modified to accept a printer device as an argument instead of performing the environment variable extraction itself.

8.3.4 rdc_Restage

rdc_Restage was designed in conjunction with rdc_Save. Both processes perform the same general function, just reversed. Because rdc_Save was coded first, many of the units that were coded for the rdc_Save function were converted to RDCS global units before implementing

rdc_Restage. Several inconsistencies still exist that need to be addressed for the sake of consistency and source code reuse. The following recommended design modifications should be made for rdc_Restage:

1. rdc_Restage does not take advantage of the functionality provided by rdc_Init. rdc_GetArgs is a self-contained function inside the rdc_Restage.c unit. This function could be eliminated entirely because rdc_Init performs this function.

2. rdc_Save makes use of a #define named RDC_RESTAGE_BIN. rdc_Restage defines a separate #define named RDC_RESTAGE_DEFAULT_BIN. RDC_RESTAGE_BIN should be used in place of RDC_RESTAGE_DEFAULT_BIN, and a call to rdc_TapeBinCount should be used in rdc_Restage to eliminate assumptions about the tape device being used.

3. rdc_Restage should add, for consistency, a -b bin option allowing the restage DLT BIN number to be overridden.

4. rdc_Restage performs an explicit call to chdir(2) to change directories. rdc_Save uses the tar(1) -C option to perform this function. rdc_Restage should be examined for removal of the chdir(2) call in favor of the tar(1) -C option.

5. rdc_Restage currently calls rdc_ShutDown to terminate the process in the event of a failure. This call should be removed and replaced with a call to lps_LogShutdownMessage(LPS_FAILURE). In addition, on successful termination, the direct call to LPS_LOGMESSAGE to report that the processing is completing should be replaced with a call to lps_LogShutdownMessage(LPS_SUCCESS).

Input

The only input to rdc_Restage is the process (command line) arguments. The format of the process arguments have been slightly modified since LPS Build 1, but are essentially unchanged.

- **-b bin** – This option does not currently exist in the rdc_Restage function, but does exist in the rdc_Save function. This option should be added to the rdc_Restage process to allow the DLT BIN number to be used for restaging to be specified on the command line. Currently, DLT BIN number 6 is required.

- **-d device** – This option is optional, but allows the tape device to be specified on the command line. If this option is not provided, the LPS_TAPE_DEV environment variable is used to obtain this information.

Output

Once restaging is successful, this process should have extracted the tape contents (both the raw wideband data file and the raw wideband data accounting file) and updated the database with the information defined in the raw wideband accounting file. In addition, the RDC_ACCT table should reflect the online state of the raw wideband data. If a record exists in the database reflecting this raw wideband data, the record is simply updated to reflect the online state. If a record does not exist reflecting this raw wideband data, a new one is created.

Notes

The LPS_TAPE_DEV and LPS_TAPE_LIBRARY_DEV environment labels are both required for rdc_Restage. Both variables are related to the DLT device, but identify different controllers. LPS_TAPE_DEV defines the tape device, and LPS_TAPE_LIBRARY_DEV defines the robotic arm that loads and unloads a tape from the DLT slots. LPS_TAPE_LIBRARY_DEV is a small computer system interface (SCSI) controller.

At the time of the Release 2 turnover to Integration and Test, the default values of LPS_TAPE_DEV and LPS_TAPE_LIBRARY_DEV were not known, therefore “unknown” was used as the default. If the default devices are known, the default values should be modified to reflect the change. Due to the currently specified defaults, if these environment variables are not set, the rdc_Restage process fails.

8.3.5 rdc_Save

rdc_Save was designed in conjunction with rdc_Restage. Both processes perform the same general function, just reversed. Because rdc_Save was coded first, many of the units that were coded for the rdc_Save function were converted to RDCS global units before implementing rdc_Restage. Several inconsistencies still exist that need to be addressed for the sake of consistency and source code reuse. The following recommended design modifications should be made for rdc_Save:

1. rdc_Save should add, for consistency, a -d device allowing command line specification of the device to use for archiving.

2. rdc_Restage created an RDCS global unit named rdc_TermSig to handle signal catching. rdc_Save generated a unit named rdc_SaveSignalHandler that performs nearly the same function. rdc_TermSig should be enhanced to be used by both rdc_Restage and rdc_Save, and the rdc_SaveSignalHandler should be removed.

3. rdc_Restage redirects the tar(1) output to a temporary file for logging purposes. rdc_Save performs the tar(1) command in QUITE mode. rdc_Save should be examined to make use of the output redirection. This may provide the end user with beneficial tar(1) feedback.

It should be noted that RDCS global rdc_SystemMonitor was generated during the implementation of rdc_Save. It was decided that the tarring of the tape would be a long process, and the user would need some feedback to ensure that the save to tape was proceeding as expected. With this in mind, rdc_SystemMonitor was designed to provide a means to perform this task, and so that it could be used for other processes. Because of the generic design of this unit, rdc_Restage also makes use of this function.

Input

The only input to the rdc_Save process is the process (command line) arguments. The format of the process arguments have been slightly modified since LPS Build 1, but are essentially unchanged.

- **-b bin** – This option is optional, but allows the default DLT BIN number for archival to be overridden. If this option is not provided, rdc_Save attempts to locate the

$LPS_TEMPFILE_PATH/lpsTapeLibraryBinFile file to obtain the BIN to be used for archiving. If this file does not exist, BIN 0 is assumed.

- **-d device** – This option currently does not exist in rdc_Save, but does exist in rdc_Restage. This option should be added to the rdc_Save process to allow the tape device to be specified on the command line. Currently, the LPS_TAPE_DEV environment variable is used to obtain this information.

- **filename** – Specifies the complete path and filename to be archived. The only validations performed on this filename are that the raw wideband data file must end in a .data extension, and the file must have an associated accounting file (.acct extension) residing in the same directory.

Output

On successful archival, this process should have saved the raw wideband data file and the raw wideband data accounting file to tape, deleted the raw wideband data files (if L0R processed), updated the archival and online state of the raw wideband data, and generated a tape label.

Notes

When the rdc_Save process has successfully completed, lpsTapeLibraryBinFile will be updated to reflect the DLT BIN number to be used for the next scheduled tape archive. This is, basically, the last function that rdc_Save performs before terminating. Therefore, lpsTapeLibraryBinFile reflects the current DLT BIN to be used for archiving on startup of the process.

The rdc_Save process will exit immediately if the BIN number read from lpsTapeLibraryBinFile (next scheduled) exceeds the number of available archiving slots (return value of rdc_TapeBinCount - 1). It was assumed that the LPS operator would have to reload the new set of tapes if this situation occurs, so this process reports an informational message to the LPS Journal and quits.

Pay careful attention to the ordering of processing when modifying this unit. rdc_GenLabel accesses lpsTapeLibraryBinFile to generate a correct tape label. If this file is updated (rdc_SetBinNumber) before rdc_GenLabel is spawned, the tape label will be incorrect.

The LPS_TAPE_DEV and LPS_TAPE_LIBRARY_DEV environment labels are both required for the rdc_Save process. Both variables are related to the DLT device, but identify different controllers. LPS_TAPE_DEV defines the tape device, and LPS_TAPE_LIBRARY_DEV defines the robotic arm that loads and unloads a tape from the DLT slots. LPS_TAPE_LIBRARY_DEV is an SCSI controller.

At the time of the Release 2 turnover to Integration and Test, the default values of LPS_TAPE_DEV and LPS_TAPE_LIBRARY_DEV were not known, so unknown was used as the default. If the default devices are known, the default values should be modified to reflect the change. Due to the currently specified defaults, if these environment variables are not set, the rdc_Save process fails.

8.3.6 rdc_Terminate

rdc_Terminate was established as an abort function to be called whenever an RDCS process needs to be aborted. This process provided a single interface definition with other subsystems (e.g., MACS) for process aborting. This allowed the method for terminating processes within the RDCS to be an internal issue so that changes to methods for terminating RDCS processes could be handled in a single place.

It is recommended that all RDCS processes be aborted using the rdc_Terminate process.

Input

The rdc_Terminate process was designed after LPS Build 1 and provided for removal of about three different executables that were designed to accomplish the same task in different manners.

- **processName** – This argument specifies the process name to be terminated. This is performed using the killall(1M) function provided by UNIX. killall(1M) was chosen because it provides simple implementation of this function, and all RDCS processes are currently terminated using the same mechanism. The use of signal handlers were implemented in all time-consuming RDCS processes requiring possible early termination.

Output

There is no output of the rdc_Terminate process other than error message logging indicating that the process did not exist.

Notes

Some RDCS processes currently do not initialize signal handlers to handle aborting. These are, generally, the processes that are not time consuming. They either perform their specified task or they do not. These processes have been flagged in this section as candidates for some redesign, e.g., incorporating rdc_Init. If the design is changed, the processes would be able to properly clean up on receipt of the termination signal.

Because this process uses killall(1M) to abort, all processes with the same name will be terminated by definition of killall(1M). For example, if two rdc_Save processes are running, both will receive the termination signal and quit. It was decided that this is a safe design because of all the RDCS processes, only rdc_DeleteFiles can have multiple running processes.

8.3.7 rdc_Transmit

rdc_Transmit is software that falls under the umbrella requirement for testing. The importance of transmitting data at 75 Mbps was the sole driver for implementing this software, therefore, many of the features that rdc_Capture provides are not provided in rdc_Transmit.

For the most part, the design of rdc_Transmit mirrors that of rdc_Capture, with the following caveats:

- No LPS database access
- No statistical output other than the transmission rate written to the log

- The input file is transmitted in its entirety unless the rdc_Transmit software is preempted by a signal. In other words, there is no duration argument such as in rdc_Capture.

See Mizar versus HPDI in Section 8.3.1 for additional details.

Input

When initially designed, the command line arguments were identical to that of rdc_Capture. Once in implementation, it was determined that some of the arguments were unnecessary.

- **-i** – See rdc_Capture
- **-s** – See rdc_Capture
- **-l** – Not needed, so it was removed
- **-b Start_Date-&-Time** – Not needed, so it was removed
- **-e Stop_Date-&-Time** – Not needed, so it was removed.
- **filename** – Specifies the complete path and filename to be transmitted. Due to limitations of the HPDI device, two 10-megabyte buffers must be filled for the transmit to work. Because of this limitation, files may only be transmitted if they are greater than or equal to 20 megabytes.

Output

The only output of the rdc_Transmit function is the serial data stream. It should be noted that a file that is not a multiple of 10 megabytes will be padded with zeros to allow the last section of data to transfer.

8.3.8 rdc_UpdRDCAcct

rdc_UpdRDCAcct was incorporated into the RDCS design to provide a means of performing some database integrity checks in the database. It was decided that the LPS database could be down for some unknown period of time while rdc_Capture continued to place raw wideband data files onto the system. In addition, during this downtime, raw wideband data files could have been moved off the system to make room for later contacts. This process identifies the raw wideband data files on the system and updates the RDC_ACCT table accordingly.

Input

rdc_UpdRDCAcct has no input.

Output

Following successful completion of rdc_UpdRDCAcct, the RDC_ACCT records existing in the table will be updated to reflect the online state of the associated raw wideband data file. Records that do not exist for online raw wideband data files will be inserted with information obtained from their associated raw wideband data accounting files.

Notes

rdc_UpdRDCAcct currently does not call rdc_Init to perform initialization. If rdc_Init were used, this process would be more consistent with the other RDCS processes, and unnecessary source code could be removed, e.g., the call to lps_db_Connect and the extraction of environment variables.

8.3.9 rdc_vmereset

rdc_vmereset is source obtained from SGI, but is not supported by SGI. This source code is required for the HPDI device to initialize correctly, so if it is not used, undetermined results could occur when capturing or transmitting data. It was decided that the source code, versus the executable, had to be delivered if operating system upgrades had to be performed on the LPS strings or porting to different platforms was required.

This process should not have to be called by the LPS operator or through any other user. rdc_Capture and rdc_Transmit fork/execute this process when the VME Bus needs to be reset.

Little is known about this source other than that it resets a VME bus.

Input

- **-a VMEbus-adap-num** – This option is required. It specifies the VME Bus adapter that is connected to the HPDI device. For LPS, it should always be 61.
- **-d reset-duration-microseconds** – This option is optional. It specifies the duration in microseconds that the process will hold down the reset line on the adapter. The default is used by LPS, and the value is 80.

Output

There is no output, other than informational messages to stdout/stderr, produced by this process.

Notes

Do not modify this source code unless absolutely sure about the pending results. If this process does not succeed in resetting the VME Bus, undetermined results could occur when capturing or transmitting data.

8.4 Raw Data Processing Subsystem

The RDPS consists of frame synchronization, CCSDS Grade 3 Services, BCH decoding, CADU annotation, generate trouble file, and provide return-link statistics of input data.

- Frame synchronization used SCLF strategy to synchronize CCSDS frame synchronization.
- CCSDS Grade 3 Services performs error detection and correction such as CRC, RS decoding. It also identifies fill CADU and virtual channel identifier (VCID) changes and annotates CADU.
  - CRC will detect the existence of error bits in the VCDU header, mission, and pointer data fields.

  - RS will protect the VCDU header and correct up to two symbols error in each RS code block.

- BCH decoding detects the errors in both mission data and data pointer fields in the CADU, capable to correct up to three error bits in each mission data or data pointer code word.

- Each CADU is annotated after each decoding stage. The filled CADU is identified and annotated. If the VCID is changed, it is annotated as well.

- Uncorrectable CADU will be stored in the trouble file for reference.

- The statistics of all CADUs received in the return link will be in the RDP_ACCT database table and complete the RDPS processing.

8.5 Major Frame Processing Subsystem

8.5.1 Data Communication

The MFPS design used following two methods to communicate data between functions:

1. Global variables – If a data item was used by more than two functions that are not in the same chain of function calls, it was made a global variable to facilitate communication. Also, a data item was made a global variable if it needed to be communicated across a deep (five or more) set of nested function calls to conserve stack space.

2. Function arguments – The data items that were not made global variables were passed on to other functions as arguments. The data items to be modified by the called function were passed as pointers. Also, large data items were passed as pointers to conserve stack space.

8.5.2 Memory Allocation

To eliminate any uncertainty about availability of memory during processing, the MFPS design does not allow dynamic allocation of memory. All memory required for MFPS processing is static and is allocated at startup.

8.6 PCD Processing Subsystem

8.6.1 Scene Identification Algorithm

The pcd program loses synchronization with the stream of unpacked PCD cycles whenever there is a gap in the VCDU sequence. This occurs even when the number of missing VCDUs is known by inference from the value of the VCDU counter. Because the number of fill words in an unpacked PCD cycle is variable, it is impossible to determine the position within the first new unpacked cycle represented by the first unpacked word after the gap. Therefore, the pcd program discards data words after a gap until unpacked cycle synchronization is established.

The pcd program loses synchronization with the stream of PCD minor frames whenever there is a gap for the same reason. All data after the gap and prior to the next minor frame synchronization code is discarded.

8.6.2 Majority Vote Failure Reporting

The pcd program counts missing data words in an unpacked PCD cycle as a majority vote failure. Data words can be missing from an unpacked PCD cycle whenever a gap in the VCDU sequence truncates the cycle. The count of majority vote failures at any level of aggregation is therefore the sum of both the number of unpacked cycles in which three data words were present but unequal and the number of unpacked cycles in which fewer than three data words were present.

8.7 Image Data Processing Subsystem

8.7.1 Band

The idp_band process is a critical function for the IDPS. If it fails, all IDPS processing will terminate. This process runs concurrently with the idp_mwd processes. It will complete ahead of the idp_browse and idp_acca processes. The main logic for band file generation is contained in the idp_BandFillFile function. Calls to the HDF-EOS API are contained in the idp_HDF module.

8.7.2 Moving Window Display

The moving window display (MWD) is an X11/Motif application that can compete with the ORACLE Forms based LPS GUI for system resources such as the X Window color palette. The result can be a black screen in the MWD. The conflict can be avoided by editing the Tk2Motif file in the user's $HOME directory and setting the "Tk2Motif*UsePrivateColormap: True" field. This causes the ORACLE Forms system to use a private color map that does not usurp the entire system color map at the expense of the MWD. If the file does not exist in the user's home directory, copy it from $ORACLE_HOME/guicommon/tk2/admin/Tk2Motif.gray.

At startup, the idp_mwd process forks a copy of itself, creating a child process. The parent process reads major frame data from the idp_band process from a named pipe and forwards the data to its child process through an internal pipe. The child MWD process interfaces with the X Window system. The MWD is split into two separate UNIX processes so that the reading of scan data and the handling of X Window events do not have to wait for each other, which could cause the system to stall.

The MWD makes use of an X11/Motif resource file, Idp_mwd (note the uppercase "I"), which should be placed in the user's $HOME directory. This file controls the appearance of the MWD form. Refer to a Motif manual for a description of resource files.

8.7.3 Browse

The idp_browse process is started when the IDPS is started; however, it remains dormant until the first subinterval is completed and the first set of band files have been produced. It reads the band files as input.

The IDPS currently relies entirely on waveletting to reduce the size of browse images. However, if a requirement is added to perform subsampling, the interface to implement this is already provided in the idp_HDFBandAccess.GetScene function. Setting the Subsampling Factor parameter in the argument list to a number "n" greater than 1 will cause the GetScene function to read every nth pixel of every nth line in the bandfile's swath.

8.7.4 Automatic Cloud Cover Assessment

The idp_acca process is started when the IDPS is started; however, it remains dormant until the first subinterval is completed and the first set of band files have been produced. It reads the band files as input.

The version of the automatic cloud cover assessment (ACCA) algorithm implemented in the current version of LPS can be ascertained with the UNIX "what" command. Type "what $IDPS_BIN/idp_acca |sort -u" at the shell command line. The result will be a list of function names and the version numbers used to build the idp_acca executable. Among the function names is a string "ACCA Algorithm ID Ver. = <version>." This corresponds to the algorithm version identifier stored in the LPS metadata.

8.7.5 Named Pipe

The idp_band and idp_mwd processes communicate with each other through an IRIX named pipe (/tmp/idp_band_mwd.pipe). idp_band sends major frame data to idp_mwd by this means because the size of the major frame records exceeds the maximum size message that can be placed on a message queue. The idp (main) process creates the pipe in the /tmp directory when IDPS initializes and removes it when IDPS processing completes.

8.7.6 Additional Debug Output

If the $IDPS_DEBUG environment variable is defined in the shell (it does not matter what it is set to), the IDPS will dump the contents of the HDF error stack to text files. The HDF error stack contains error messages generated by HDF library functions while attempting to read or write HDF files. If error messages occur in the LPS Journal relating to the generation of band or browse files and additional information is needed to diagnose the problem, the HDF error stack might be useful. The idp_band process writes HDF errors to the /tmp/idp_band.hdf.errs file and the idp_browse process to the /tmp/idp_browse.hdf.errs file.

The IDPS will write additional status messages to the LPS Journal if the DEBUG symbol is defined during compilation. It does not need to be set to anything. Adding -DDEBUG to the compiler options in a Makefile will turn on this extended status message logging. These status messages are intended for debugging only and are excessive during normal LPS processing.

8.8 LPS Data Transfer Subsystem

8.8.1 senddan

The LDTS has the capability to open multiple sessions to connect to the ECS server, and each session can contain multiple DANs. However, the senddan program is designed to create one DAN only and send it over to ECS in an open session. This is based on a small number of contact periods captured in 1 day for LPS and makes the design of the senddan program simpler and easier to maintain. In case it is necessary to send multiple DANs to ECS at the same time, the senddan program can be invoked multiple times simultaneously by the MACS and has no impact on the senddan program.

When a failed DAN is resent to ECS, a new DAN with a new DAN sequence number is generated. The failed DAN will be marked as "canceled DAN" for good. For resending the suspended DANs, the same DANs will be sent over to ECS.

If an error encountered in the TCP/IP socket or the crucial information cannot be retrieved from the database, the senddan program will be aborted. When this happens, it requires the LPS operator to troubleshoot the problem.

8.8.2 rcvddn

To keep the design simple, rcvddn will only receive one DDN message for an open connection between LPS and ECS. If multiple DDN messages are sent from ECS simultaneously, the incoming DDN messages will be queued by the TCP/IP socket and the rcvddn program will process them seriously based on the order of their presence in the socket.

If an error encountered in the TCP/IP socket or the crucial information cannot be retrieved from the database, the rcvddn program will be aborted. When this happens, it requires the LPS operator to troubleshoot the problem. However, if the error will only affect the database consistency and does not prevent the rcvddn program from continuing, the program will keep running and an alert message will be logged to the LPS Journal.

Because rcvddn is a server process and continues checking the socket for any incoming message, a health message is printed on the console periodically to inform the LPS operator of the existence of the server.

If the data exchange sequence is compromised due to system error, the same DDN may be received twice. In this case, rcvddn will only process the DDN once. However, to guarantee ECS receives the data delivery acknowledgment (DDA) message, rcvddn will send the DDA messages twice.

8.8.3 stopddn

The stopddn program is designed to be used by the MACS to terminate the rcvddn server process via user interface. To prevent the operator from terminating the rcvddn server prematurely while rcvddn is still processing a DDN message, the stopddn program first checks the rcvddn processing status. If a DDN message is currently being processed, stopddn will wait until the rcvddn server completes the processing and then terminate the rcvddn server process.

8.8.4 deletefile

If a contact period’s L0R files are marked for retention, the files cannot be deleted automatically or manually. To delete the L0R files manually, the retention mark must be cleared. Once the L0R files are deleted, all successful database transactions will be committed even if one or more database transaction error(s) is encountered.

8.8.5 LDTS Database Tables

To keep the file transfer status for the contact period L0R files, an LDT_File_Set_Info record and corresponding LDT_DAN_Info record are created. They are associated with each other by the contact sequence ID and the file version number. The LDT_File_Set_Info record is created by the MACS when LPS L0R processing begins and is maintained by the LDTS. When a failed DAN is resent, the recreated new DAN and the failed DAN point to the same LDT_File_Set_Info record to maintain LDTS database integrity.

For LDT_File_Set_Info record, one or more subinterval LDT_File_Group_Info record(s) is created and is associated with each other by the subinterval sequence ID. Likewise, one or more LPS_File_Info record(s) is created for a LDT_File_Group_Info record and is associated with the LDT_File_Group_Info record by the subinterval sequence ID.

Section 9. Other Gotchas

9.1 LPS Message Logging Mechanism

LPS uses the UNIX syslogd daemon to log the error and status messages into the LPS Journal file that is configured in /etc/syslog.conf. Currently, LPS is configured to select “local0” facility at all severity level messages except the kernel messages. The pathname for the destination action file, LPS_Journal, should be configured to match $LPS_JOURNAL_PATH. The following severity levels are defined and used by LPS:

- LPS_MSG_EMERG – system is unusable, Emergency Message
- LPS_MSG_ALERT – action must be taken immediately, Alert Message
- LPS_MSG_CRIT – critical conditions, Critical Message
- LPS_MSG_ERR – error conditions, Error Message
- LPS_MSG_WARNING – warning conditions, Warning Message
- LPS_MSG_NOTICE – normal but significant condition, Notification Message
- LPS_MSG_INFO – informational, Informational Message
- LPS_MSG_DEBUG – debug-level messages, Debug Message

When a message is ready to go into the LPS Journal, the subsystem calls lps_LogMessage.c with the message severity level, source filename, source line number where the message is generated, and message string. The message is reformatted according to the following format:

date + priority_level + hostname + “:” + GID + “[“ + source_unit_name + “:” + source_line_number + ”]” + message_string

The reformatted message is then logged into the LPS_Journal by syslogd.

If the messages need to be printed on the console for debugging purpose, set the $LPS_LOG_STDOUT environment variable.

9.2 Raw Data Capture Subsystem

The RDCS is currently inconsistent with the rest of LPS with respect to the executable names. It uses mixed case executable names, whereas the rest of LPS uses all lowercase names for the executables.

The rdc_vmereset function was source code obtained by SGI. This source code is NOT supported by SGI, but is required on the LPS for initializing the HPDI device. Failure to use this executable could produce undeterministic results when using the HPDI device and is not recommended. This source should be carefully considered before upgrading the operating system or porting the software to another platform.

All RDCS processes make use of the same environment variable (RDC_STATUS_INTERVAL) to determine the reporting period for informational message logging. This implementation may

cause difficulty if the tape functions need a different default reporting period than the capture and transmit functions. It may be beneficial to define unique environment variables for each process, or each function type, to eliminate potential aggravation for the LPS operator.

There are no additional “gotchas” for the RDCS not previously discussed in the previous sections.

9.3 PCD Processing Subsystem

9.3.1 Subinterval Change Lag in Scene Identification

For a given iteration of PCD’s outermost loop, all functions normally process data from the same subinterval, even when they are not processing the same data. However, when a subinterval changes, the scene identification functions (pcd_MainDetermineScenes and its subordinates) will process saved data from the previous subinterval in the current iteration and save the data from the new subinterval for processing in the next iteration.

The pcd program design involves a significant amount of global-scope processing state information. Some important state information are the current subinterval and the subinterval change flag, pcd_subIntvChangedFlag. Unfortunately, there are a variety of ways in which the current subinterval is accessed from the global state information available. The most popular method is through the variable pcd_subIntvList. It lists a summary for each subinterval encountered and includes a count field that can be used as an index. The pcd_subintv_index global variable contains the same information.

It is vital to remember that for pcd_MainDetermineScenes and its subordinates, global subinterval state information does not apply when the subinterval changes. In particular, using pcd_subintv_index or pcd_subIntvList.PCD_Sub_Intv_Count to access information about the subinterval to which the data being processed belongs will fail. The data being processed actually belongs to the previous subinterval. Also, pcd_subIntvChangedFlag will be true when the saved data from the last subinterval is being processed and not for the first data from the new subinterval.

For pcd_MainDetermineScenes and its subordinates, pcd_cycleFromNewSubIntv should be used to determine whether or not the data being processed is from a new subinterval.

Acronyms

ACCA	automatic cloud cover assessment
API	application programmatic interface
ASCII	American Standard Code for Information Interchange
BCH	Bose-Chaudhuri-Hocquenghem
BSD	Berkeley Software Distribution
CADU	channel access data unit
COTS	commercial off-the-shelf
CCSDS	Consultative Committee for Space Data Systems
CRC	cyclic redundancy check
DAN	data availability notice
DBAR	database access routine
DBMS	database management system
DDA	data delivery acknowledgment
DDF	Data Distribution Facility
DDN	data delivery notice
DLT	digital linear tape
ECS	EOS Core System
EDC	EROS Data Center
EOS	Earth Observation System
EROS	Earth Resources Observation System
ETM+	Enhanced Thematic Mapper Plus
FIFO	first-in-first-out
FILO	first-in-last-out
FSP	frame synchronization process
GHA	Greenwich hour angle
GOTS	Government off-the-shelf
GSFC	Goddard Space Flight Center
GUI	graphical user interface

HDF	hierarchical data format
HPDI	high-speed peripherel device interface
IAS	Image Assessment System
ICD	interface control document
IDD	interface definition document
IDPS	image data processing subsystem
I/O	input/output
IP	Internet Protocol
IPC	interprocess communication
JPL	Jet Propulsion Laboratory
LDTS	LPS data transfer subsystem
LGS	Landsat Ground Station
LPS	Landsat 7 Processing System
MACS	management and control subsystem
MFPS	major frame processing subsystem
MOC	Mission Operations Center
MO&DSD	Mission Operations and Data Systems Directorate
MSCD	mirror scan correction data
MWD	moving window display
NASA	National Aeronautics and Space Administration
NCSA	National Center for Supercomputing Applications
PCD	payload correction data
PCDS	PCD processing subsystem
PID	process identifier
PVL	Parameter Value Language
Q&A	quality and accounting
RDCS	raw data capture subsystem
RDPS	raw data processing subsystem
RS	Reed-Solomon

RSL	Reusable Software Library
SCLF	search, check, lock, and flywheel
SCSI	small computer system interface
SGI	Silicon Graphics, Inc.
SQL	Structured Query Language
TCP	Transmission Control Protocol
VCDU	virtual channel data unit
VCID	virtual channel identifier
VME	Versa-Module European